Bổ túc toán

Nội dung chính : Trong chương này, chúng ta sẽ nhắc lại một cách khái quát các thuật ngữ và kiến thức toán học sẽ được dùng đến trong suốt giáo trình. Đó là các kiến thức liên quan đến đồ thị, cây, tập hợp, quan hệ và một vài phương pháp chứng minh toán học thông thường. Nếu các khái niệm này là mới đối với bạn, bạn nên xem lại một cách cẩn thận. Ngược lại, nếu chúng không là mới, bạn có thể đọc lướt nhanh qua chương này, nhưng hãy chắc chắn rằng mình đã nắm rõ về chúng. Mục tiêu cần đạt : Sau chương này, sinh viên có thể : Xác định tập hợp và các phép toán cơ bản trên tập hợp Định nghĩa một quan hệ, lớp quan hệ và các tính chất của quan hệ. Xác định quan hệ tương đương và phép bao đóng. Chứng minh một phát biểu toán học theo phương pháp quy nạp. Nắm vững các khái niệm về đồ thị và cây. Kiến thức cơ bản : Các kiến thức Toán có liên quan. Tài liệu tham khảo : [1] John E. Hopcroft, Jeffrey D.Ullman – Introduction to Automata Theory, Languages and Computation – Addison – Wesley Publishing Company, Inc – 1979 (trang 1 – trang 12). [2] V.J. Rayward-Smith – A First course in Formal Language Theory (Second Editor) – McGraw-Hill Book Company Europe – 1995 (Chapter 1: Mathematical Prerequisites) [3] Các giáo trình về Toán rời rạc

TẬP HỢP (Sets)

Một tập hợp là tập các đối tượng không có sự lặp lại. Mỗi đối tượng trong tập hợp được gọi là phần tử (element) của tập hợp đó.

Ký hiệu tập hợp

Nếu số phần tử trong một tập hợp không quá lớn, hay nói cách khác – tập hợp là hữu hạn, tập hợp có thể được đặc tả bằng cách liệt kê các phần tử của nó.

Thí dụ 1.1 : D xác định tập hợp các ngày trong tuần :

D = { Mon, Tues, Wed, Thurs, Fri, Sat, Sun }

Các phần tử trong tập hợp viết cách nhau bởi dấu “, “ và đặt trong cặp dấu { và }. Không có sự bắt buộc về thứ tự liệt kê các phần tử trong tập hợp. Chẳng hạn, tập hợp D cũng tương đương với tập hợp sau :

D = { Mon, Wed, Fri, Thurs, Sun, Tues, Sat }

Nếu phần tử x là thành phần của tập hợp A, ta viết x   A (đọc là x thuộc A), và nếu x không là phần tử của A, ta viết x   A (đọc là x không thuộc A). Chẳng hạn : Mon   D nhưng Kippers   D.

Nếu một tập hợp chứa một số khá lớn các phần tử hay thậm chí là một số vô hạn, người ta có thể không liệt kê tất cả các phần tử mà đặc tả tập hợp theo một số tính chất đặc trưng của nó.

Thí dụ 1.2 : D = { x   x là một ngày trong tuần }

P = { y   y là số nguyên tố }

X = { x   x > 2 }

Mọi tập hợp đều chứa các phần tử thuộc vào một không gian xác định nào đó, ký hiệu là U. Không gian tương ứng có thể được định nghĩa là một tập số nguyên, số thực, …

Một trường hợp đặc biệt của tập hợp là tập hợp rỗng (empty set). Tập hợp này không có chứa bất kỳ phần tử nào, ký hiệu bởi   hoặc { }.

Ta nói tập hợp A là tập hợp con (subset) của tập hợp B khi mọi phần tử của A là thành phần của B (ký hiệu A   B). Ngược lại, A không là tập con của B (A   B).

Thí dụ 1.3 : { 1, 2, 4 }   { 1, 2, 3, 4, 5 } nhưng { 2, 4, 6 }   { 1, 2, 3, 4, 5 }

Có thể suy ra rằng tập hợp A   U và   A,   A

Hai tập hợp A và B được gọi là bằng nhau (A = B), khi A   B và B   A

Thí dụ 1.4 : { 1, 2, 3, 4 } = { 2, 1, 4, 3 } nhưng { 1, 2, 3, 4 }   { 2, 1, 3, 5 }

Tập hợp tất cả các tập hợp con của tập A được gọi là tập lũy thừa (power set) của A và xác định bởi 2A.

Thí dụ 1.5 : Giả sử A = { 1, 2, 3 }

Thì 2A = {   , {1 }, {2 }, {3}, {1, 2}, {2, 3}, {3, 1}, {1, 2, 3} }

Các phép toán trên tập hợp

Các toán tử cơ bản trên tập hợp bao gồm các toán tử một ngôi (unary) và hai ngôi (binary) như sau :

1) Phép phần bù (complement) : A' = {x   x   A }

2) Phép hợp (union) : A   B = {x   x A hoặc x B}

3) Phép giao (intersection) : A   B = {x   x A và x B}

4) Phép trừ (difference) : A \ B = {x   x A nhưng x B}

5) Tích Đecac : A   B = {(a,b)   a A và b B}

Thí dụ 1.6 : Cho A = {1, 2} và B = {2, 3}

Ta có : A   B = {1, 2, 3}

A   B = {2}

A \ B = {1}

A   B = {(1, 2), (1, 3), (2, 2), (2, 3)}

2A = { , {1}, {2}, {1, 2}}

Lưu ý : Nếu A và B lần lượt có số phần tử là n và m thì tập hợp A   B có n   m phần tử và tập 2A có 2n phần tử.

-

QUAN HỆ (Relations)

Cho hai tập hợp A và B. Một quan hệ hai ngôi R giữa A và B là tập hợp chứa tất cả các tập hợp con của A   B mà thành phần thứ nhất A được gọi là miền xác định (domain) của R, còn B gọi là miền giá trị (range) của R (có thể trùng với miền xác định). Chúng ta sẽ thường dùng quan hệ hai ngôi mà miền xác định và miền giá trị cùng thuộc một tập hợp S nào đó. Trong trường hợp này, ta gọi đây là một quan hệ trên S. Nếu R là một quan hệ và (a,b) là một cặp trong R thì ta viết aRb.

Thí dụ 1.7 : Cho S = { 0, 1, 2, 3}

. Quan hệ "thứ tự nhỏ hơn" trên S được xác định bởi tập :

L = {(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)}

. Quan hệ "bằng" trên S được xác định bởi tập :

E = {(0, 0), (1, 1), (2, 2), (3, 3)}

. Quan hệ "chẵn lẻ" trên S được xác định bởi tập :

P = {(0, 0), (1, 1), (2, 2), (3, 3), (0, 2), (2, 0), (1, 3), (3, 1)}

Các tính chất của quan hệ

Ta gọi một quan hệ R trên tập S là:

Phản xạ (reflexive) : nếu aRa là đúng a   S

Đối xứng (symmetric) : nếu aRb thì bRa

Bắc cầu (transitive) : nếu aRb và bRc thì aRc

Thí dụ 1.8 :

. L không là quan hệ phản xạ trên S vì (0, 0)   L, nhưng E và P là 2 quan hệ mang tính phản xạ.

. L không là quan hệ đối xứng trên S vì (0, 1)   L nhưng (1, 0)   L, tuy nhiên cả E và P đều mang tính đối xứng.

. Cả L, E và P đều là các quan hệ mang tính bắc cầu, nhưng X = {(1, 0), (0, 3)} thì không vì (1, 3)   X.

Quan hệ tương đương

Một quan hệ R trên tập S có đủ các tính chất phản xạ, đối xứng và bắc cầu được gọi là quan hệ tương đương.

Thí dụ 1.9 :E và P là các quan hệ tương đương, còn L và X không là các quan hệ tương đương trên S.

Một tính chất quan trọng của quan hệ tương đương là nếu R là quan hệ tương đương trên tập S thì R phân hoạch tập S thành các lớp tương đương (equivalence class) Si không rỗng và rời nhau, tức là S = S1   S2 ... và với mọi i   j ta có :

+ Si Ç Sj = Æ

+ Với mỗi a,b cùng thuộc Si thì aRb là đúng.

+ Với mỗi a   Si và b   Sj thì aRb là sai.

Lưu ý rằng số lớp tương đương có thể là vô hạn. Vậy nếu R là quan hệ tương trên S và a   S, ta có :

Si = [a] = {b Î S   aRb}

Thí dụ 1.10 :

. E có 4 lớp tương đương khác nhau: [0] = {0}, [1] = {1}, [2] = {2} và [3] = {3}

. P có 2 lớp tương đương khác nhau: [0] = [2] = {0, 2} và [1] = [3] = {1, 3}

Bao đóng của quan hệ

Giả sử P là tập hợp một số tính chất của các quan hệ, bao đóng P (P - closure) của một quan hệ R trên tập S là quan hệ nhỏ nhất có chứa tất cả các cặp của R thoả mãn các tính chất trong P.

Bao đóng bắc cầu R+ của R được xác định như sau :

i) Nếu (a,b) thuộc R thì (a,b) thuộc R+.

ii) Nếu (a,b) thuộc R+ và (b,c) cũng thuộc R thì (a,c) thuộc R+.

iii) Không còn gì thêm trong R+.

Bao đóng phản xạ và bắc cầu R* của R được xác định như sau :

R* = R+   {(a, a)   a   S}

Thí dụ 1.11 :Cho quan hệ R = {(1, 2), (2, 2), (2, 3)} trên tập hợp S = {1, 2, 3}

Khi đó ta có :

R+ = {(1, 2), (2, 2), (2, 3), (1, 3)}

R* = {(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)}

PHÉP CHỨNG MINH QUY NẠP

Phần lớn các định lý trong giáo trình sẽ được chứng minh bằng phương pháp quy nạp toán học :

Giả sử ta cần chứng minh một mệnh đề P(n) với n là một số nguyên không âm. Nguyên lý quy nạp toán học cho P(n) được chứng minh theo 2 bước như sau :

i) P (0) , và

ii) P(n - 1) kéo theo P (n),   n   1.

Bước (i) được gọi là cơ sở quy nạp, bước (ii) được gọi là bước quy nạp với P(n-1) là giả thiết quy nạp.

Thí dụ 1.12 :Dùng quy nạp, chứng minh biểu thức :

$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2\text{n}+1)}{6}$

Cơ sở quy nạp : Thay n = 0 trong vế phải của biểu thức và nhận thấy cả 2 vế đều bằng 0   P (0) luôn đúng.

Bước quy nạp : Thay n bởi n - 1 để có được giả thiết quy nạp P(n-1), sau đó tìm cách để chứng minh P(n), tức chứng minh   n   1, ta có :

$\sum_{\text{i=0}}^{\text{n - 1}} i^2 = \frac{\text{(n - 1)n(2n -1)}}{6} \Rightarrow \sum_{\text{i=0}}^{n} i^2 = \frac{\text{n}(n+1)(2\text{n}+1)}{6}$

Ta có nhận xét rằng :

$\sum_{\text{i=0}}^{n} i^2 = \sum_{\text{i=0}}^{\text{n - 1}} i^2 + n^2$

$\frac{\text{(n -1)n(2n -1)}}{6} + n^2 = \frac{\text{n(n+1)(2n+1)}}{6}$ Vậy nếu ta vận dụng giả thiết quy nạp thì chỉ còn phải chứng minh biểu thức :

Với một vài phép biến đổi đại số đơn giản, biểu thức trên có thể được chứng minh dễ dàng. Hay P(n) được chứng minh,   n.

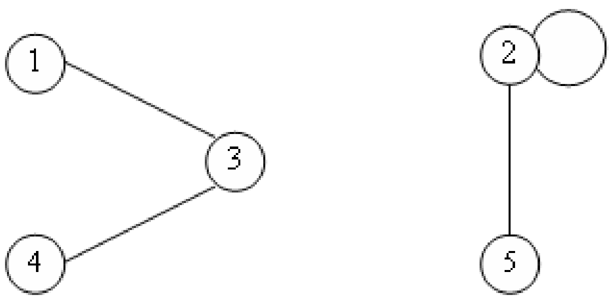
ĐỒ THỊ VÀ CÂY

Đồ thị (Graph)

Một đồ thị, ký hiệu G = (V, E), bao gồm một tập hữu hạn các đỉnh V (còn gọi là nút) và một tập các cạnh E nối giữa 2 nút.

Thí dụ 1.13 :Đồ thị cho bởi :V = {1, 2, 3, 4, 5}

và E = {(n, m)   n + m = 4 hoặc n + m = 7}



Hình 1.1 - Ví dụ về đồ thị

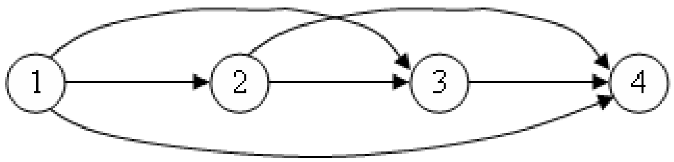
Một đường đi (path) trên một đồ thị là dãy các đỉnh v1, v2 , . . ., vk, k   1, sao cho trong đó có một cạnh (vi ,vi +1) cho mỗi i, 1   i < k. Độ dài của đường đi là k - 1. Nếu v1 = vk thì đường đi là một chu trình.

Chẳng hạn : 1, 3, 4 là một đường đi trong đồ thị trên.

Đồ thị có hướng (directed graph)

Một đồ thị có hướng cũng là dạng đồ thị được xác định bởi G = (V, E), trong đó V là tập các đỉnh, còn E là tập các đỉnh có thứ tự gọi là các cung (hay các đường nối có hướng giữa 2 đỉnh). Ký hiệu một cung từ v đến w có dạng v   w.

Thí dụ 1.14 : Đồ thị có hướng G = ({1, 2, 3, 4 }, { i   j   i < j })



Hình 1.2 - Một đồ thị có hướng

Một đường đi trên một đồ thị có hướng là dãy các đỉnh v1, v2 , . . ., vk, k 1, sao cho với mỗi i, 1 i < k, có một cung từ vi đến vi +1. Chẳng hạn 1 2 3 4 là một đường đi trên đồ thị định hướng trên (từ 1 đến 4).

Cây (trees)

Cây (cây định hướng có thứ tự) là một đồ thị có hướng với các tính chất sau :

i) Có một nút đỉnh gọi là nút gốc

ii) Mỗi nút còn lại đều được dẫn ra từ một nút cha ở trên nó :

- Các nút có dẫn ra nút con sau nó được gọi là nút trung gian hay nút trong.

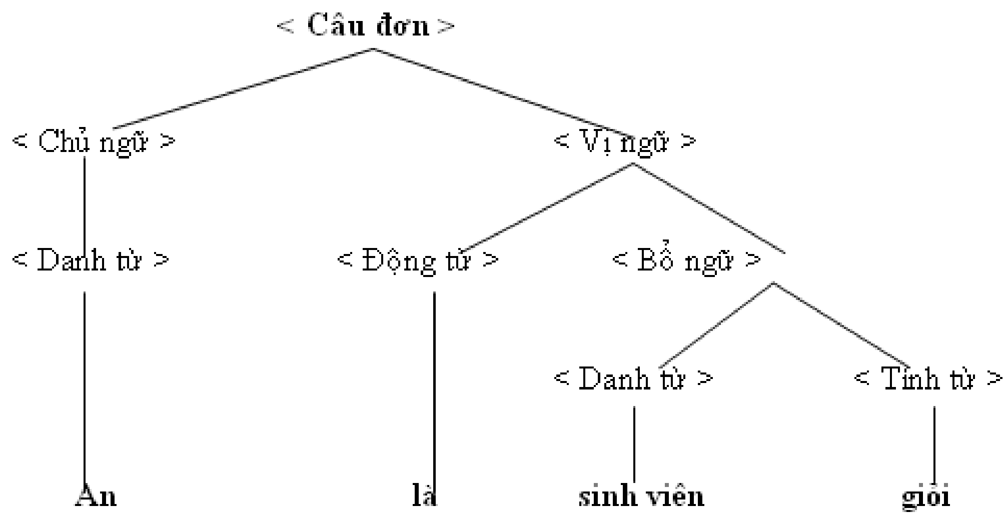
- Các nút không dẫn ra nút con gọi là nút lá.

iii) Thứ tự duyệt trên cây là từ trái sang phải.

Trong một cây, người ta thường dùng các khái niệm nút cha và nút con để lần lượt chỉ thứ tự trước và sau của sự phát sinh nút từ nút gốc trên cây. Nếu có một đường đi từ nút v1 đến nút v2 thì v1 được gọi là nút cha của v2 và ngược lại, v2 sẽ là nút con của nút v1.

Ta thường vẽ cây với nút gốc ở trên cùng và các cung chỉ xuống phía dưới, do vậy các ký hiệu mũi tên trở nên không còn cần thiết nữa. Các nút con của mỗi nút trên cây sẽ được vẽ lần lượt từ trái qua phải theo thứ tự đã xác định.

Thí dụ 1.15 : Cây minh họa cấu trúc cú pháp của một câu đơn trong ngôn ngữ tiếng Việt "An là sinh viên giỏi"



Hình 1.3 - Cây minh họa một câu đơn

BÀI TẬP CHƯƠNG I

1.1. Nếu không gian tập hợp là tập các số nguyên dương nhỏ hơn 20. Hãy viết rõ các phần tử trong các tập hợp được xác định như sau :

1. { x   x + 2 < 10}
2. { x   x là số nguyên tố }
3. { x   x = x2}
4. { x   2x = 1}
5. { x   3x < 20}

1.2. Cho tập hợp S = {0, 1, 2, 3, 4, 5, 6}

Hãy viết rõ các phần tử trong các tập hợp được xác định như sau :

1. { x   x   S và x chẵn }
2. { x   x   S và x   x2 + 1 }

1.3. Cho A = {0, 1, 2} và B = {0, 3, 4}. Hãy viết rõ các tập hợp sau :

A B ; A B ; A \ B ; A x B và 2A

1.4. Cho ví dụ về quan hệ :

a) Phản xạ và đối xứng, nhưng không bắc cầu.

b) Phản xạ và bắc cầu, nhưng không đối xứng.

c) Đối xứng và bắc cầu, nhưng không phản xạ.

Trong mỗi trường hợp trên, chỉ rõ tập hợp trên đó quan hệ được xác định.

1.5. Chứng minh các quan hệ sau đây là các quan hệ tương đương và cho các lớp tương đương của chúng.

a) Quan hệ R1 trên các số nguyên định nghĩa bởi : iR1j khi và chỉ khi i = j.

b) Quan hệ R2 trên một tập thể người định nghĩa bởi : pR2q khi và chỉ khi p, q sinh cùng ngày và cùng năm.

1.6. Cho tập hữu hạn A. Hãy tìm những quan hệ tương đương trên A có số các lớp tương đương là lớn nhất hay nhỏ nhất.

1.7. Cho hai tập hợp sau A = {2, 3, 4, 5} và B = {1, 3, 5, 7, 9}. Giả sử R là quan hệ :

R = {(x, y) A B x < y}

Hãy liệt kê các cặp quan hệ thứ tự trong R.

1.8. Tìm bao đóng bắc cầu, bao đóng phản xạ và bắc cầu của quan hệ được cho như sau trên S = { 1, 2, 3, 4, 5}:

{(1, 2), (2, 3), (3, 4), (5, 4)}

1.9. Cho S = {0, 1, 2} và R = {(0, 1), (1, 2)}. Tìm R* và R+.

Ôtômát tuyến tính giới nội và văn phạm cảm ngữ cảnh
Nội dung chính : Trong chương này, chúng ta xét thêm một loại ôtômát, không mạnh bằng máy Turing, được gọi là ôtômát tuyến tính giới nội (Linear Bounded Automata – LBA). Đồng thời cũng xét thêm lớp văn phạm tương ứng với nó, là lớp văn phạm L1 hay còn gọi là văn phạm cảm ngữ cảnh, lớp văn phạm nằm giữa lớp văn phạm L0 và văn phạm phi ngữ cảnh L2. Từ đó ta hoàn thành sự phân cấp các ngôn ngữ thành 4 cấp, gọi là sự phân cấp Chomsky. Mục tiêu cần đạt: Cuối chương, sinh viên cần phải nắm vững: Khái niệm LBA, định nghĩa và các thành phần. Sự tương đương giữa LBA và văn phạm cảm ngữ cảnh. Mối tương quan giữa các lớp ngôn ngữ. Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, sinh viên cần hiểu rõ các dạng ôtômát đã được giới thiệu trong các chương trước, đặc biệt là mô hình máy Turing; nắm vững cơ cấu các lớp văn phạm… Tài liệu tham khảo : [1] Nguyễn Văn Ba – Giáo trình ngôn ngữ hình thức – Trường Đại học Bách khoa Hà nội – 1994. [2] A. C. Fleck - Context Sensitive Languages: http://www.cs.uiowa.edu/~fleck/PartIIIxpar [3] Linear Bounder Automata: http://cs.engr.uky.edu/~lewis/texts/theory/automata/lb-auto.pdf

ÔTÔMÁT TUYẾN TÍNH GIỚI NỘI (LBA)

Ta gọi Ôtômát tuyến tính giới nội (Linear Bounded Automata - LBA) là một máy Turing không đơn định và không có khả năng nới rộng vùng làm việc ra khỏi mút trái và mút phải của chuỗi nhập. Nó phải thỏa hai điều kiện sau :

1) Bộ chữ cái nhập của nó có chứa thêm hai ký hiệu đặc biệt và $ dùng làm ký hiệu đánh dấu mút trái và mút phải.

2) LBA không thực hiện phép chuyển sang trái (L) từ và không thực hiện phép chuyển sang phải (R) từ $, và cũng không viết các ký hiệu khác lên và $.

LBA đơn giản là một máy Turing nhưng thay vì sử dụng một băng không giới hạn cho việc tính toán, nó bị hạn chế chỉ trong phạm vi băng chứa chuỗi nhập x với hai ô chứa các ký hiệu đánh dấu cận đầu mút. Sự giới

hạn này làm cho việc tính toán phải thông qua một số các hàm tuyến tính trên độ dài chuỗi, do đó ta gọi mô hình này là ôtômát tuyến tính giới nội. LBA không dùng các ô trống ở trên băng về phía trái và phía phải của chuỗi nhập, vì vậy ký hiệu khoảng trắng B (Blank) như đã dùng ở máy Turing là không cần dùng ở đây. Trái lại, để LBA nhận biết được giới hạn bên trái và giới hạn bên phải của chuỗi nhập, ta phải đưa thêm vào bộ chữ cái nhập hai ký hiệu đặc biệt , $ để đánh dấu mút trái và mút phải của chuỗi. Vậy, tại thời điểm bắt đầu, chuỗi nhập đưa vào ở trên băng sẽ có dạng w $, trong đó w (- { , $})* là chuỗi cần đoán nhận. Trong quá trình làm việc, khi đầu đọc đọc tới ô có chứa hay $, thì phép chuyển tiếp theo sau đó chỉ có thể là đổi trạng thái, chuyển đầu đọc trở lại phía trong phạm vi băng (tức chuyển sang phải khi gặp và chuyển sang trái khi gặp $), và không được phép viết ký hiệu gì khác trên băng tại ô đang đọc khi gặp và $.

Định nghĩa LBA

Một cách hình thức, LBA là một hệ thống M(Q, , , ,qo, , $, F), trong đó các thành phần Q, , , qo, F vẫn như đã định nghĩa ở máy Turing, còn , $ và hàm chuyển :

: Q (Q { L, R})

phải thỏa mãn điều kiện:

- Nếu (p, Y, E) (q,) thì Y = và E = R
- Nếu (p, Y, E) (q, $) thì Y = $ và E = L

Ngôn ngữ được chấp nhận bởi LBA

Ta định nghĩa ngôn ngữ L(M) được đoán nhận bởi LBA M là tập hợp :

L(M) = { w w (- { , $})* và qo w$ ⊢M* q với q F và * }

Chú ý rằng các ký hiệu đánh dấu hai đầu mút ngay từ hình thái bắt đầu chúng đã có mặt trên băng nhập, nhưng chúng không được xem như thuộc một phần của chuỗi được chấp nhận hay không được chấp nhận

bởi LBA. Vì đầu đọc của LBA không thể dịch chuyển ra ngoài phần chuỗi nhập nên chúng ta không cần định nghĩa các khoảng trống (ký hiệu Blank) phía bên phải của $.

VĂN PHẠM CẢM NGỮ CẢNH (CSG)

Ta gọi văn phạm cảm ngữ cảnh (Context Sensitive Grammar - CSG) là một hệ thống G (V, T, P, S), trong đó:

1. V là một tập hữu hạn các biến hay ký hiệu không kết thúc.
2. T là một tập hữu hạn các ký hiệu cuối, V T =
3. P là tập hữu hạn các luật sinh dạng trong đó , (V T)*, chuỗi phải có chứa biến và ràng buộc
4. S V là ký hiệu bắt đầu.

Ta định nghĩa ngôn ngữ do văn phạm cảm ngữ cảnh G sinh ra là

L(G) = { w w * và S * w}

L(G) được gọi là ngôn ngữ cảm ngữ cảnh (Context Sensitive Language - CSL). Thuật ngữ "cảm ngữ cảnh" có xuất xứ từ một dạng chuẩn của văn phạm dạng này, trong đó mỗi luật sinh có dạng 1A 2 1 2 với , cho thấy một biến A chỉ có thể được thay thế bởi một chuỗi (khác rỗng) trong "ngữ cảnh" 1 - 2. Điều đó không giống như trong văn phạm phi ngữ cảnh, với các luật sinh có dạng A (0), sự thay thế này không đòi hỏi ngữ cảnh.

Thí dụ 8.1 : Xét CSG G (V, T, P, S) với V ={ S, B, C}, ={a, b, c} và P gồm các luật sinh như sau :

1) S aSBC

2) S aBC

3) CB BC

4) aB ab

5) bB   bb

6) bC   bc

7) cC   cc

Một cách phi hình thức, bằng cách áp dụng một số luật sinh cho các chuỗi dẫn xuất sinh ra ngôn ngữ, ta dễ thấy rằng văn phạm G sinh ra ngôn ngữ có dạng :

L = {anbncn n   1}

Thật vậy, với luật sinh (1) và (2) ta có chuỗi dẫn xuất S * an(BC)n. Sau đó, bằng cách áp dụng luật sinh (3), mọi biến B sẽ được thay thế lên trước các biến C trong chuỗi dẫn xuất : an(BC) * anBnCn. Bởi luật sinh (4) và (5), mọi biến B sẽ được thay thế thành các ký hiệu kết thúc b, và cuối cùng với (6) và (7), mọi biến C cũng sẽ được thay thế thành c. Tóm lại, ta có chuỗi dẫn xuất như sau :

S * an(BC)n * anBnCn * anbncn

Bài toán thành viên với CSG (Membership)

ĐỊNH LÝ 8.1 : Tồn tại giải thuật để xác định với mọi ngôn ngữ cảm ngữ cảnh CSG G(V, T, P, S) bất kỳ và một chuỗi nhập w   T*, liệu chuỗi w có thuộc ngôn ngữ L(G) hay không.

Chứng minh

Giả sử   w   = n. Ta lập đồ thị mà mỗi đỉnh là một chuỗi thuộc (V   T)* có độ dài nhỏ hơn hoặc bằng n, có một cung từ đỉnh   đến đỉnh   nếu   G . Như vậy một đường trong đồ thị đó tương ứng với một suy dẫn trong G. Vậy w   L(G) khi và chỉ khi có một đường đi từ đỉnh bắt đầu S tới đỉnh w trong đồ thị. Dùng bất cứ giải thuật nào cho phép tìm đường nối hai đỉnh trong đồ thị (đã có nhiều thuật toán như thế), ta sẽ xác định được phải chăng đã có đường đi từ đỉnh S tới đỉnh w.

Thí dụ 8.2 : Xét CSG G (V, T, P, S) với các luật sinh được cho như trong Thí dụ 8.1 trên và xét chuỗi nhập w = abbc. Ta cần xác định xem liệu chuỗi w L(G)?

Để tìm đường đi từ đỉnh S tới đỉnh abbc trong đồ thị nói trên ta có thể dùng phương pháp "vết dầu loang" như sau:

Lập các R(i), i = 0, 1, 2, … theo quy tắc sau:

R(0) = { S }

R(i) = R(i -1) { với R(i -1) và w }

Do R(0) R(1) … R(i) R(i +1) … tập các đỉnh, vậy tồn tại số k nào đó sao cho:

R(k) = R(k +1) = R(k +2) = …

Do đó quá trình thành lập các R(i) sẽ có thể ngừng sau k bước.

Và w L(G) khi và chỉ khi có i k để cho w R(i).

Trong thí dụ trên, giả sử khi ta xét w = 4, ta có:

R(0) = { S }

R(1) = {S, aSBC, aBC}

R(2) = {S, aSBC, aBC, abC}

R(3) = {S, aSBC, aBC, abC, abc}

R(4) = R(3)

Vậy chuỗi abbc không thuộc L(G).

SỰ TƯƠNG ĐƯƠNG GIỮA LBA VÀ CSG

Chúng ta chú ý rằng LBA có thể chấp nhận các chuỗi rỗng , còn CSG không thể sinh ra chuỗi rỗng. Ngoài trường hợp đó ra thì LBA sẽ chấp nhận chính xác tất cả các chuỗi được sinh ra từ CSG.

ĐỊNH LÝ 8.2 : Nếu L là một CSG thì L sẽ được chấp nhận bởi một LBA nào đó.

Chứng minh

Cách chứng minh định lý này cũng tương tự như cách chứng minh của định lý 7.9 ở chương trước về sự tương đương giữa lớp ngôn ngữ sinh từ văn phạm loại 0 với lớp ngôn ngữ mà máy Turing chấp nhận, chỉ khác là ở đây không cần dùng một băng nhập thứ hai để phát sinh các dạng câu theo chuỗi dẫn xuất lần lượt theo các suy dẫn của văn phạm, mà chỉ cần dùng rãnh thứ hai trên băng nhập của LBA vào việc đó.

Cho G = (V, T, P, S) là một CSG, ta xây dựng ôtômát LBA M như sau: Băng nhập của LBA gồm hai rãnh : rãnh 1 chứa chuỗi nhập w với các ký hiệu đánh dấu , $ ở hai đầu, rãnh 2 dùng để phát sinh các dạng câu . Trạng thái bắt đầu, nếu w = thì M ngừng và không chấp nhận input, nếu không thì đầu đọc viết ký hiệu S ở rãnh 2, ngay dưới ký hiệu bên trái nhất của chuỗi w, tiếp đó M thực hiện quá trình sau:

1) Chọn trong số không đơn định một chuỗi con của chuỗi trên rãnh 2 sao cho là một luật sinh trong P.

2) Thay bởi , nếu cần thiết ta phải dịch chuyển phần cuối chuỗi sang phải cho đủ chỗ, tuy nhiên nếu dịch chuyển ra ngoài $ thì LBA ngừng và không chấp nhận.

3) (Hình thái hiện tại ở rãnh 1 là w $, còn ở rãnh 2 là chuỗi , mà S G và w). So sánh rãnh 1 và rãnh 2, nếu = w thì LBA ngừng và chấp nhận w. Nếu không thì trở về bước (1).

Như vậy khi M chấp nhận chuỗi w, thì S G* w. Ngược lại nếu S G* w thì mọi dạng câu xuất hiện trong chuỗi dẫn xuất đó đều thoả mãn w , bởi vì mọi luật sinh trong văn phạm G đều thỏa .

Như vậy M có thể thực hiện chuỗi dẫn xuất đó trên rãnh 2, giữa hai ký hiệu đánh dấu đầu mút và $. Vậy M chấp nhận chuỗi nhập w.

Tóm lại M sẽ chấp nhận mọi chuỗi sinh ra bởi văn phạm G.

ĐỊNH LÝ 8.3 : Nếu L = L(M) với một LBA M (Q, , , ,qo, , $, F) thì L – { } là một ngôn ngữ cảm ngữ cảnh.

Chứng minh

Cách chứng minh định lý này cũng tương tự như cách chứng minh của định lý 7.10 ở chương trước, bằng cách ta xây dựng một CSG G thực hiện 3 giai đoạn:

- Giai đoạn 1: Văn phạm cho phép sinh ra một chuỗi w (chuỗi nhập của M), cũng được chứa trong , $ và q0.

- Giai đoạn 2: Văn phạm lặp lại công việc của M.

- Giai đoạn 3: Khi xuất hiện trạng thái q F, ta thu về chuỗi w với lưu ý rằng các luật sinh đều có = .

Quá trình mô phỏng lại các luật sinh đó bởi các luật sinh của CSG sẽ không có gì vướng mắc. Chỉ ở giai đoạn 3, việc xoá đi các ký hiệu đánh dấu hai đầu mút và $, q không được phép làm rút ngắn chuỗi nhập lại. Để giải quyết vướng mắc này, ta gắn các ký hiệu , $, q kề bên với các ký hiệu của chuỗi nhập mà không để đứng rời ra như trước.

Cụ thể, giai đoạn 1 thực hiện bởi các luật sinh trong G sau:

S1 [a, q0 a]S2S1 [a, q0 a$]

S2 [a, a]S2,S2 [a,a$]

a - { , $}

Các luật sinh trong G cho phép thực hiện giai đoạn 2, giống như LBA M thực hiện (sinh viên tự xây dựng xem như bài tập).

Cuối cùng, ở giai đoạn 3, các luật sinh sau đây sẽ được sử dụng, với q F :

[a, q] a

a - { , $} và , có thể có.

Chú ý rằng số luật sinh là hữu hạn, vì và / hoặc chỉ gồm , $ và một ký hiệu nhập vào. Chúng ta cũng có thể xoá thành phần thứ hai của một biến nếu nó liền kề với ký hiệu kết thúc bằng cách dùng các luật sinh dạng:

[a,]b ab

b[a,] ba

a, b - { , $} và có thể có.

Như vậy các luật sinh vừa được xây dựng mô tả văn phạm là CSG và có thể chứng minh L(M) - { } = L(G).

TƯƠNG QUAN GIỮA CÁC LỚP NGÔN NGỮ

Ngôn ngữ đoán nhận bởi các văn phạm cũng được phân loại theo tên của từng lớp văn phạm, ta gọi đó là sự phân cấp Chomsky về ngôn ngữ.

Có 4 lớp ngôn ngữ đã được giới thiệu – tập đệ quy liệt kê (r.e), ngôn ngữ cảm ngữ cảnh (CSL), ngôn ngữ phi ngữ cảnh (CFL) và tập chính quy (r) tương đương với 4 lớp ngôn ngữ loại 0, 1, 2 và 3.

Theo lý thuyết được xây dựng xuyên suốt trong giáo trình này, ta có thể tóm tắt lại như sau:

a) L là ngôn ngữ loại 0 khi và chỉ khi L được đoán nhận bởi một máy Turing.

b) L là ngôn ngữ loại 1 khi và chỉ khi L được đoán nhận bởi một ôtômát tuyến tính giới nội (sai khác chuỗi rỗng)

c) L là ngôn ngữ loại 2 khi và chỉ khi L được đoán nhận bởi một ôtômát đẩy xuống (không đơn định).

d) L là ngôn ngữ loại 3 khi và chỉ khi L được đoán nhận bởi một ôtômát hữu hạn (sai khác chuỗi rỗng).

Ta cũng cần lưu ý rằng sự phân cấp ngôn ngữ như trên là một bao hàm thức nghiêm ngặt, thể hiện quy luật sau:

a) Lớp các ngôn ngữ loại 3 là tập con thực sự của lớp ngôn ngữ loại 2. Thật vậy mọi văn phạm chính quy đều là văn phạm phi ngữ cảnh. Hơn nữa người ta có thể chứng minh rằng ngôn ngữ {0n1n n 1} là một ngôn ngữ phi ngữ cảnh, nhưng không phải là ngôn ngữ chính quy.

b) Lớp các ngôn ngữ loại 2 không chứa các chuỗi rỗng là tập con thực sự của lớp ngôn ngữ loại 1. Thật vậy mọi văn phạm phi ngữ cảnh có dạng chuẩn Chomsky đều là văn phạm cảm ngữ cảnh. Hơn nữa người ta có thể chứng minh rằng ngôn ngữ {a 2i i 1} là ngôn ngữ cảm ngữ cảnh nhưng không là ngôn ngữ phi ngữ cảnh.

c) Lớp các ngôn ngữ loại 1 là tập con thực sự của lớp các ngôn ngữ loại 0. Thật vậy, mọi văn phạm cảm ngữ cảnh đều là văn phạm cấu trúc không hạn chế. Mặt khác người ta cũng đề xuất được những ngôn ngữ là đệ quy liệt kê (loại 0), mà không cần làm ngữ cảnh (loại 1). Các thí dụ đó được xây dựng dựa trên các khái niệm “đệ quy” và “sự giải được”, mà khuôn khổ giáo trình này không cho phép đề cập đến.

Tổng kết chương VIII: Với sự giới thiệu mô hình ôtômát tuyến tính giới nội LBA và lớp ngôn ngữ cảm ngữ cảnh mà nó đoán nhận, mô hình phân cấp ngôn ngữ theo Noam Chomsky đã được hoàn chỉnh.

BÀI TẬP CHƯƠNG VIII

8.1. Xây dựng văn phạm cảm ngữ cảnh sinh ra các ngôn ngữ sau:

a) { ww | w Î (0+1)+}

b) { 0k | k = i2 và i ³ 1}

c) { 0i | i không là số nguyên tố}

d) { ai b2i c3i   i ³ 1}

e) { ai bi ck   i ³ 1, k   1}

8.2. Thiết kế ôtômát tuyến tính giới nội LBA đoán nhận các ngôn ngữ sau:

1. { an bn cn   n ³ 1}

b) { ww | w Î (a + b + c)*}

Ôtômat hữu hạn và biểu thức chính quy
Nội dung chính: Trong chương này, ta sẽ nghiên cứu một loại "máy trừu tượng" gọi là ôtômát hữu hạn. Chúng là công cụ dùng đoán nhận một lớp ngôn ngữ khá đơn giản gọi là lớp ngôn ngữ chính quy. Trước hết, hai dạng của ôtômát hữu hạn sẽ lần lượt được trình bày và có sự chứng minh rằng chúng tương đương nhau về khả năng đoán nhận ngôn ngữ. Tiếp đó, ta sẽ đề cập đến biểu thức chính quy - một phương tiện khác để xác định ngôn ngữ và ta lại thấy rằng lớp ngôn ngữ do các ôtômát hữu hạn chấp nhận chính là lớp ngôn ngữ chính quy. Phần tiếp theo của chương sẽ đề cập đến mối quan hệ giữa cơ chế ôtômát và các biểu thức chính quy dùng ký hiệu cho ngôn ngữ. Cuối chương này, một vài ứng dụng cụ thể của ôtômát hữu hạn sẽ được trình bày. Mục tiêu cần đạt: Kết thúc chương này, sinh viên cần nắm vững : Khái niệm ôtômát hữu hạn, các thành phần, các dạng và sự khác biệt cơ bản giữa hai dạng. Cách thức chuyển đổi tương đương từ dạng đơn định sang không đơn định và ngược lại. Viết biểu thức chính quy ký hiệu cho tập ngôn ngữ chính quy. Mối liên quan giữa ôtômát hữu hạn và biểu thức chính quy. Vẽ sơ đồ chuyển trạng thái (đơn định hoặc không đơn định) từ một biểu thức chính quy. Tìm các ứng dụng thực tế khác từ mô hình ôtômát hữu hạn. Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, sinh viên cần có một số các kiến thức liên quan về lý thuyết đồ thị, lý thuyết mạch; hiểu các khái niệm cơ bản về kiến trúc máy tính; có sử dụng qua một số trình soạn thảo văn bản thông thường … Tài liệu tham khảo : [1] John E. Hopcroft, Jeffrey D.Ullman – Introduction to Automata Theory, Languages and Computation – Addison – Wesley Publishing Company, Inc – 1979 (Chapter 2 : Finite Automata and Regular Expressions). [2] Phan Thị Tươi – Trình biên dịch – Nhà xuất bản Giáo dục – 1986 (Chương 3 : Bộ phân tích từ vựng). [3] J.A.Garcia and S.Moral- Theory of Finite Automata : http://decsai.ugr.es/~jags/fat.html [4] Donald R. Biggar - Regular Expression Matching Using Finite Automata: http://www3.sympatico.ca/dbiggar/FA.home.html

ÔTÔMÁT HỮU HẠN (FA : Finite Automata)

Ôtômát hữu hạn FA là một mô hình tính toán của hệ thống với sự mô tả bởi các input và output. Tại mỗi thời điểm, hệ thống có thể được xác

định ở một trong số hữu hạn các cấu hình nội bộ gọi là các trạng thái (states). Mỗi trạng thái của hệ thống thể hiện sự tóm tắt các thông tin liên quan đến những input đã chuyển qua và xác định các phép chuyển kế tiếp trên dãy input tiếp theo.

Trong khoa học máy tính, ta có thể tìm thấy nhiều ví dụ về hệ thống trạng thái hữu hạn, và lý thuyết về ôtômát hữu hạn là một công cụ thiết kế hữu ích cho các hệ thống này. Chẳng hạn, một hệ chuyển mạch như bộ điều khiển (Control Unit) trong máy tính. Một chuyển mạch thì bao gồm một số hữu hạn các cổng (gate) input, mỗi cổng có 2 giá trị 0 hoặc 1. Các giá trị đầu vào này sẽ xác định 2 mức điện thế khác nhau ở cổng output. Mỗi trạng thái của một mạng chuyển mạch với n cổng bất kỳ sẽ là một trường hợp trong 2n phép gán của 0 và 1 đối với các cổng khác nhau. Các chuyển mạch thì được thiết kế theo cách này, vì thế chúng có thể được xem như hệ thống trạng thái hữu hạn. Các chương trình sử dụng thông thường, chẳng hạn trình sọan thảo văn bản hay bộ phân tích từ vựng trong trình biên dịch máy tính cũng được thiết kế như các hệ thống trạng thái hữu hạn. Ví dụ bộ phân tích từ vựng sẽ quét qua tất cả các dòng ký tự của chương trình máy tính để tìm nhóm các chuỗi ký tự tương ứng với một tên biến, hằng số, từ khóa, …Trong quá trình xử lý này, bộ phân tích từ vựng cần phải nhớ một số hữu hạn thông tin như các ký tự bắt đầu hình thành những chuỗi từ khóa. Lý thuyết về ôtômát hữu hạn thường được dùng đến nhiều cho việc thiết kế các công cụ xử lý chuỗi hiệu quả.

Máy tính cũng có thể được xem như một hệ thống trạng thái hữu hạn. Trạng thái hiện thời của bộ xử lý trung tâm, bộ nhớ trong và các thiết bị lưu trữ phụ ở mỗi thời điểm bất kỳ là một trong những số rất lớn và hữu hạn của số trạng thái. Bộ não con người cũng là một hệ thống trạng thái hữu hạn, vì số các tế bào thần kinh hay gọi là neurons là số có giới hạn, nhiều nhất có thể là 235.

Lý do quan trọng nhất cho việc nghiên cứu các hệ thống trạng thái hữu hạn là tính tự nhiên của khái niệm và khả năng ứng dụng đa dạng trong nhiều lĩnh vực thực tế. Ôtômát hữu hạn (FA) được chia thành 2 loại: đơn định (DFA) và không đơn định (NFA). Cả hai loại ôtômát hữu hạn đều có

khả năng nhận dạng chính xác tập chính quy. Ôtômát hữu hạn đơn định có khả năng nhận dạng ngôn ngữ dễ dàng hơn ôtômát hữu hạn không đơn định, nhưng thay vào đó thông thường kích thước của nó lại lớn hơn so với ôtômát hữu hạn không đơn định tương đương.

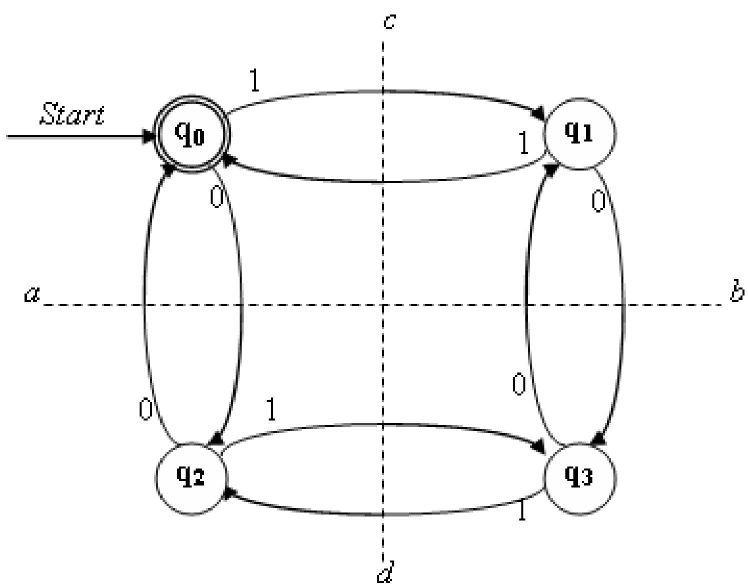
Ôtômát hữu hạn đơn định - DFA (Deterministic Finite Automata)

Một ôtômát hữu hạn đơn định (DFA) - gọi tắt là FA -gồm một tập hữu hạn các trạng thái và một tập các phép chuyển từ trạng thái này tới trạng thái khác trên các ký hiệu nhập (input symbols) được chọn từ một bộ chữ cái   nào đó. Mỗi ký hiệu nhập có đúng một phép chuyển khỏi mỗi trạng thái (có thể chuyển trở về chính nó). Một trạng thái, thường ký hiệu là q0, gọi là trạng thái bắt đầu (trạng thái ôtômát bắt đầu). Một số trạng thái được thiết kế như là các trạng thái kết thúc hay trạng thái chấp nhận.

Một đồ thị có hướng, gọi là sơ đồ chuyển (transition diagram) tương ứng với một DFA như sau: các đỉnh của đồ thị là các trạng thái của DFA; nếu có một đường chuyển từ trạng thái q đến trạng thái p trên input a thì có một cung nhãn a chuyển từ trạng thái q đến trạng thái p trong sơ đồ chuyển. DFA chấp nhận một chuỗi x nếu như tồn tại dãy các phép chuyển tương ứng trên mỗi ký hiệu của x dẫn từ trạng thái bắt đầu đến một trong những trạng thái kết thúc.

Chẳng hạn, sơ đồ chuyển của một DFA được mô tả trong hình 3.1. Trạng thái khởi đầu q0 được chỉ bằng mũi tên có nhãn "Start". Chỉ có duy nhất một trạng thái kết thúc, cũng là q0 trong trường hợp này, được chỉ ra bằng hai vòng tròn. Ôtômát này chấp nhận tất cả các chuỗi số 0 và số 1 với số số 0 và số số 1 là số chẵn.

Thí dụ 3.1 :



Hình 3.1 - Sơ đồ chuyển của một DFA

Một điều cần lưu ý, DFA sử dụng mỗi trạng thái của nó để giữ chỉ một phần của chuỗi số 0 và 1 chứ không phải chứa một số thực sự, vì thế DFA cần dùng một số hữu hạn trạng thái.

Định nghĩa

Một cách hình thức ta định nghĩa ôtômát hữu hạn là bộ gồm năm thành phần (Q, , , q0, F), trong đó :

. Q là tập hợp hữu hạn các trạng thái.

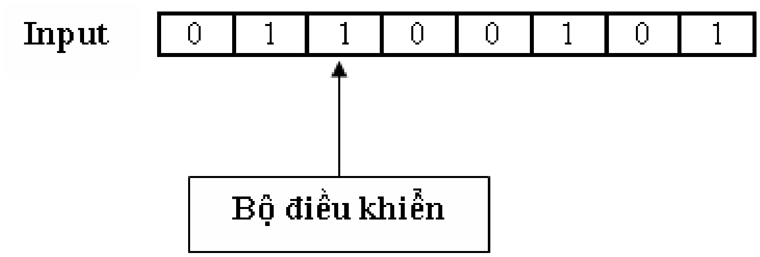
. là bộ chữ cái nhập hữu hạn.

. là hàm chuyển ánh xạ từ Q Q, tức là (q, a) là một trạng thái được cho bởi phép chuyển từ trạng thái q trên ký hiệu nhập a.

. q0 Q là trạng thái bắt đầu

. F Q là tập các trạng thái kết thúc.

Ta vẽ DFA như là bộ điều khiển hữu hạn, với mỗi trạng thái thuộc Q, DFA đọc một chuỗi các ký hiệu a từ   viết trên băng (như hình vẽ).



Hình 3.2 - Mô tả một DFA

Trong một lần chuyển, DFA đang ở trạng thái q đọc ký hiệu nhập a trên băng, chuyển sang trạng thái được xác định bởi hàm chuyển (q, a), rồi dịch đầu đọc sang phải một ký tự. Nếu (q, a) chuyển đến một trong những trạng thái kết thúc thì DFA chấp nhận chuỗi được viết trên băng input phía trước đầu đọc, nhưng không bao gồm ký tự tại vị trí đầu đọc vừa dịch chuyển đến. Trong trường hợp đầu đọc đã dịch đến cuối chuỗi trên băng, thì DFA mới chấp nhận toàn bộ chuỗi trên băng.

Hàm chuyển trạng thái mở rộng

Để có thể mô tả một cách hình thức hoạt động của một DFA trên chuỗi, ta mở rộng hàm chuyển   để áp dụng đối với một trạng thái trên chuỗi hơn là một trạng thái trên từng ký hiệu. Ta định nghĩa hàm chuyển   như một ánh xạ từ Q   *   Q với ý nghĩa (q, w) là trạng thái DFA chuyển đến từ trạng thái q trên chuỗi w. Một cách hình thức, ta định nghĩa :

1. d (q,) = q
2. (q, wa) = ((q, w), a), với mọi chuỗi w và ký hiệu nhập a.

Một số quy ước về ký hiệu :

- Q là tập các trạng thái. Ký hiệu q và p (có hoặc không có chỉ số) là các trạng thái, q0 là trạng thái bắt đầu.

- là bộ chữ cái nhập. Ký hiệu a, b (có hoặc không có chỉ số) và các chữ số là các ký hiệu nhập.
- là hàm chuyển.
- F là tập các trạng thái kết thúc.
- w, x, y và z (có hoặc không có chỉ số) là các chuỗi ký hiệu nhập.

Ngôn ngữ được chấp nhận bởi DFA

Một chuỗi w được chấp nhập bởi ôtômát hữu hạn M (Q, , , q0, F) nếu (q0, w) = p với p F. Ngôn ngữ được chấp nhận bởi M, ký hiệu L(M) là tập hợp:

L(M) = { w (q0, w) F }

Thí dụ 3.2 : Xét sơ đồ chuyển ở hình 3.1. Theo khái niệm hình thức, ta có DFA được xác định bởi M (Q, , , q0, F) với Q = {q0, q1, q2, q3}, = {0, 1}, F = {q0} và hàm chuyển như sau:

d	Inputs	
Trạng thái	0	1
q0	q2	q1
q1	q3	q0
q2	q0	q3
q3	q1	q2

Giả sử chuỗi w = 110101 được nhập vào M.

Ta có (q0, 1) = q1 và (q1, 1) = q0 ,vậy (q0, 11) = ((q0,1),1) = (q1, 1) = q0.

Tiếp tục (q0, 0) = q2, vậy (q0, 110) = ((q0, 11), 0) = q2.

Tiếp tục ta có (q0, 1101) = q3, (q0, 11010) = q1

Và cuối cùng (q0, 110101) = q0 F.

(Hay d(q0, 110101) = d(q1, 10101) = d(q0, 0101) = d(q2, 101) = d(q3, 01) = d(q1, 1) = q0 F)

Vậy 110101 thuộc L(M). Ta có thể chứng minh rằng L(M) là tập mọi chuỗi có số chẵn số 0 và số chẵn số 1.

Theo mô tả DFA như trên, ta thấy cũng có thể dùng bảng hàm chuyển (transition table) để mô tả các phép chuyển trạng thái của một ôtômát hữu hạn. Trong bảng hàm chuyển, hàng chứa các trạng thái thuộc tập trạng thái của ôtômát và cột là các ký hiệu thuộc bộ chữ cái nhập. Bảng hàm chuyển gợi ý cho chúng ta một cấu trúc dữ liệu để mô tả cho một ôtômát hữu hạn, đồng thời cũng cho thấy có thể dễ dàng mô phỏng hoạt động của DFA thông qua một chương trình máy tính, chẳng hạn dùng cấu trúc vòng lặp.

Giải thuật mô phỏng hoạt động của một DFA

. Input : Chuỗi nhập x kết thúc bởi $. Output : Câu trả lời "YES" nếu DFA chấp nhận chuỗi x và "NO" nếu ngược lại.. Giải thuật :q := q0 ;c := nextchar ; { c là ký hiệu nhập được đọc tiếp theo }While c < > $ dobeginq := d(q, c);c := nextchar ;endIf q in F then write ("YES") else write("NO");

Nhận xét :

Một cách tổng quát, ta thấy tập Q của DFA thể hiện các trạng thái lưu trữ của ôtômát trong quá trình đoán nhận ngôn ngữ, và như vậy khả năng lưu trữ của ôtômát là hữu hạn. Mặt khác, hàm chuyển d là hàm toàn phần và đơn trị, cho nên các bước chuyển của ôtômát luôn luôn được xác định

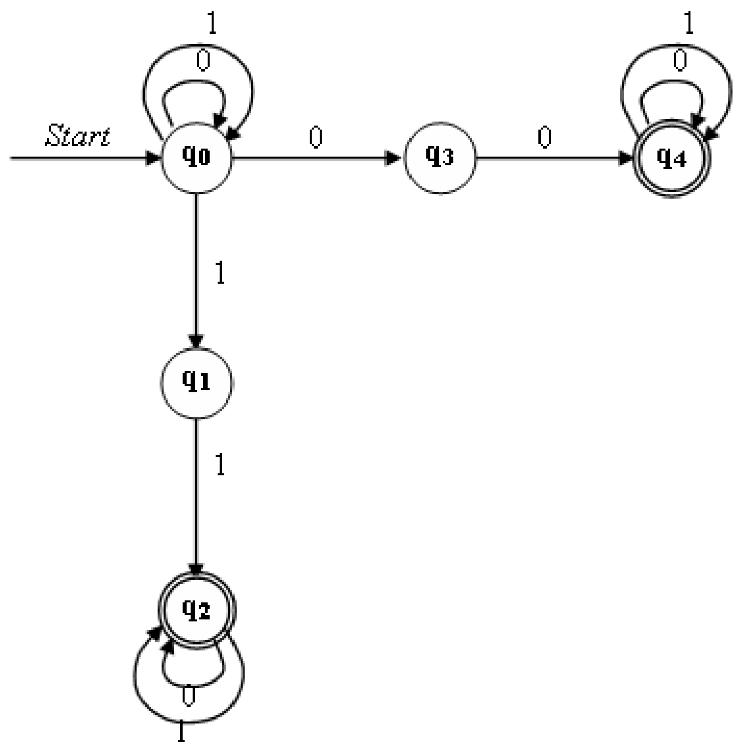
một cách duy nhất. Chính vì hai đặc điểm này mà DFA mô tả như trên được gọi là ôtômát hữu hạn đơn định.

Ôtômát hữu hạn không đơn định - NFA (Nondeterministic Finite Automata)

Xét một dạng sửa đổi mô hình DFA để chấp nhận không, một hoặc nhiều hơn một phép chuyển từ một trạng thái trên cùng một ký hiệu nhập. Mô hình mới này gọi là ôtômát hữu hạn không đơn định (NFA).

Một chuỗi ký hiệu nhập a1 a2 ... an được chấp nhận bởi một NFA nếu có tồn tại một chuỗi các phép chuyển, tương ứng với chuỗi nhập, từ trạng thái bắt đầu đến trạng thái kết thúc. Chẳng hạn, chuỗi 01001 được chấp nhận bởi ôtômát trong hình dưới đây vì có chuỗi phép chuyển qua các trạng thái q0, q0, q0, q3, q4, q4 có nhãn tương ứng là 0, 1, 0, 0, 1. NFA này chấp nhận tất cả các chuỗi có hai số 0 liên tiếp hoặc hai số 1 liên tiếp.

Thí dụ 3.3 :

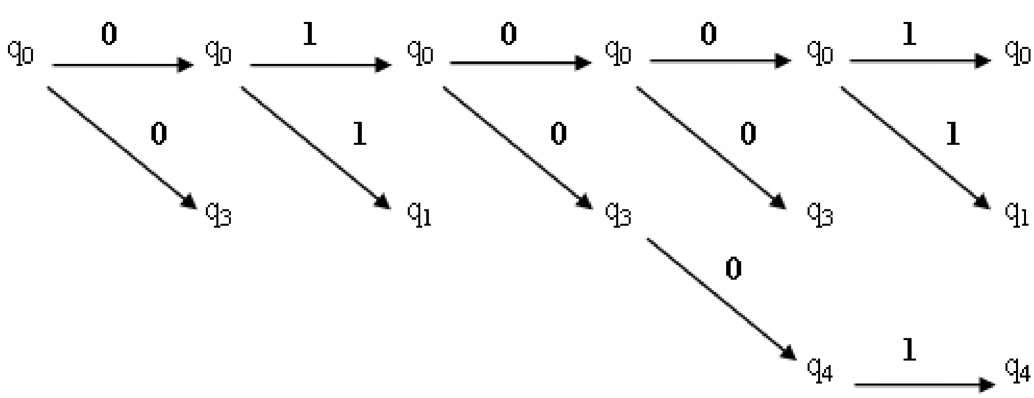


Hình 3.3 - Sơ đồ chuyển của một NFA

Chú ý rằng có thể xem ôtômát hữu hạn đơn định - DFA (hay gọi tắt là FA) là một trường hợp đặc biệt của NFA, trong đó mỗi trạng thái chỉ có duy nhất một phép chuyển trên mỗi ký hiệu nhập. Vì thế trong DFA, với một chuỗi nhập w và trạng thái q, chỉ có đúng một đường đi nhãn w bắt đầu từ q. Để xác định chuỗi w có được chấp nhận bởi DFA hay không chỉ cần kiểm tra đường đi này. Nhưng đối với NFA, có thể có nhiều đường đi có nhãn là w, và do đó tất cả phải được kiểm tra để thấy có hay không có đường đi tới trạng thái kết thúc.

Tương tự như DFA, NFA cũng hoạt động với một bộ điều khiển hữu hạn đọc trên băng nhập. Tuy nhiên, tại mỗi thời điểm, bộ điều khiển có thể chứa một số bất kỳ trạng thái. Khi có sự lựa chọn trạng thái kế tiếp, chẳng hạn như từ trạng thái q0 trên ký hiệu nhập 0 ở hình 3.3, ta phải tưởng tượng như có các bản sao của ôtômát đang thực hiện đồng thời. Mỗi trạng thái kế tiếp mà ôtômát có thể chuyển đến sẽ tương ứng với một bản sao của ôtômát mà tại đó bộ điều khiển đang chứa trạng thái đó.

Chẳng hạn, với chuỗi 01001, ta có :



Định nghĩa

Một cách hình thức ta định nghĩa ôtômát hữu hạn không đơn định NFA là một bộ 5 thành phần (Q, , , q0, F) trong đó Q, , q0 và F có ý nghĩa như trong DFA, nhưng   là hàm chuyển ánh xạ từ Q   2Q.

Khái niệm (q, a) là tập hợp tất cả các trạng thái p sao cho có phép chuyển trên nhãn a từ trạng thái q tới p.

Hàm chuyển trạng thái mở rộng

Để thuận tiện trong việc mô tả hoạt động ôtômát trên chuỗi, ta mở rộng hàm chuyển   ánh xạ từ Q   *   2Q như sau :

1. d(q, e) = {q}

1. (q, wa) = { p   có một trạng thái r trong (q, w) mà p thuộc (r, a)}

= ((q, w), a)

1. d(P, w) = Èq   P d(q, w) , P   Q.

Ngôn ngữ được chấp nhận bởi NFA

Ngôn ngữ L(M), với M là ôtômát hữu hạn không đơn định NFA (Q,   ,   , q0, F) là tập hợp :

L(M) = {w   (q0, w) có chứa một trạng thái trong F }

Thí dụ 3.4 : Xét sơ đồ chuyển của hình 3.3. Theo khái niệm hình thức, ta có :

NFA M ({q0, q1, q2, q3, q4}, {0, 1}, d, q0, {q2, q4}) với hàm chuyển d như sau :

d	Inputs	
Trạng thái	0	1
q0	{q0,q3}	{q0,q1}
q1	Æ	{q2}
q2	{q2}	{q2}
q3	{q4}	Æ
q4	{q4}	{q4}

Xét chuỗi nhập w = 01001

Ta có :d (q0, 0) = {q0, q3}

d (q0, 01) = d(d(q0, 0),1) = d({q0, q3},1) = d (q0, 1) È d (q3, 1) = {q0, q1}

Tương tự , ta có thể tính :

d (q0, 010) = {q0, q3}

d (q0, 0100) = {q0, q3, q4}

và d (q0, 01001) = {q0, q1, q4}

Do q4   F nên w   L (M).

Câu hỏi :

?

1. Hãy cho nhận xét về điểm khác biệt quan trọng giữa DFA và NFA ?
2. Theo bạn, dạng đơn định hay không đơn định sẽ dùng nhận dạng một chuỗi

dễ dàng hơn ?

Sự tương đương giữa DFA và NFA

Vì mỗi DFA là một NFA, nên rõ ràng lớp ngôn ngữ được chấp nhận bởi NFA cũng bao gồm các tập chính quy (đây chính là ngôn ngữ được chấp nhận bởi DFA). Tuy nhiên, không có cơ sở để nói rằng NFA chỉ chấp nhận duy nhất các tập hợp này. Điều đó cho thấy DFA có thể mô phỏng được hoạt động của NFA, nghĩa là với mỗi NFA, ta có thể xây dựng một DFA tương đương (chấp nhận cùng một ngôn ngữ với nó). Đặt một DFA mô phỏng hoạt động của NFA là cho phép các trạng thái của DFA tương ứng với tập các trạng thái của NFA. Tại mỗi thời điểm, DFA lưu giữ trong bộ điều khiển tất cả các trạng thái mà NFA có thể chuyển đến khi đọc cùng một input như DFA.

ĐỊNH LÝ 3.1 : Nếu L là tập được chấp nhận bởi một NFA thì tồn tại một DFA chấp nhận L.

Chứng minh

Đặt M (Q, , , q0, F) là NFA chấp nhận L.

Ta xây dựng DFA M' (Q’, , ’, q0’, F’) tương đương như sau:

- Các trạng thái của M’ là tất cả các tập hợp con của tập trạng thái của M, hay Q’= 2Q. Tại mỗi thời điểm, M’ sẽ lưu giữ trong trạng thái của nó tất cả các trạng thái có thể của M. Một phần tử trong Q’ được ký hiệu là [q1, q2,..., qi], trong đó các trạng thái q1, q2,..., qi Q. Ta xem [q1, q2,..., qi] là một trạng thái đơn của DFA tương ứng với một tập trạng thái của NFA.

- q0’ = [q0].

- F' là tập hợp các trạng thái của Q’ có chứa ít nhất một trạng thái kết thúc trong tập F của M.

- Ta định nghĩa hàm chuyển ’ như sau :

d’ ([q1, q2,..., qi], a) = [p1, p2,..., pj] nếu và chỉ nếu ({q1, q2,..., qi }, a) = {p1, p2,..., pj}

Bây giờ ta chứng minh quy nạp theo độ dài của chuỗi nhập x rằng:

d’(q0’, x) = [q1, q2,..., qi] Û d(q0, x) = {q1, q2,..., qi} (1)

Với x = 0 , ta có x = và q0’ = [q0] nên (1) hiển nhiên đúng

Giả sử (1) đúng với các chuỗi nhập có độ dài tới m.

Xét chuỗi nhập có độ dài m + 1, đặt chuỗi này là xa với a , ta có :

d’(q0’, xa) = d’(d’(q0’, x), a)

Theo định nghĩa :

d’([p1, p2,..., pi], a) = [r1, r2,..., rk] Û d({p1, p2,..., pj}, a) = {r1, r2,..., rk}.

Mặt khác theo giả thiết quy nạp d’(q0’, x) = [p1, p2,..., pj] Û d(q0, x) = {p1, p2,..., pj}, nên thay vào ta có : d’(q0’, xa) = [r1, r2,..., rk] Û d(q0, xa) = {r1, r2,..., rk}.

Dễ thấy rằng d’(q0’, x)   F' khi và chỉ khi d(q0, x) có chứa ít nhất một trạng thái   F.

Vậy L(M) = L(M’)

Vì NFA và DFA chấp nhận cùng các tập hợp, nên ta sẽ không phân biệt chúng trừ khi điều đó thật sự cần thiết, sẽ đơn giản hơn để hiểu cả hai cùng là ôtômát đơn định.

Thí dụ 3.5 : Cho NFA M ({q0, q1}, {0, 1}, d, q0, {q1}) với hàm chuyển d như sau :

d(q0, 0) = {q0, q1},d(q0,1) = {q1},d(q1, 0) = , d(q1, 1) = {q0, q1}

Ta xây dựng DFA tương đương M’ (Q’, {0, 1}, d’, [q0], F’) chấp nhận L(M) như sau :

. Q’ : chứa tất cả các tập con của {q0, q1}, vậy Q’ = {[q0], [q1], [q0, q1], }

. Hàm chuyển d’ :

Vì d(q0, 0) = {q0, q1} nên d’([q0], 0) = [q0, q1]

Tương tự : d’([q0], 1) = [q1]

d’([q1], 0) =

d’([q1], 1) = [q0, q1]

Mặt khác : d’(, 0) = d’(, 1) =

Cuối cùng : d’([q0, q1],0) = [q0, q1]

(vì d({q0, q1},0) = d(q0, 0)   d(q1, 0) = {q0, q1}   = {q0, q1})

d’([q0, q1], 1) = [q0, q1]

(vì d({q0, q1},1) = d(q0, 1)   d(q1, 1) = {q1}   {q0, q1} = {q0, q1})

. Tập trạng thái kết thúc F' = {[q1], [q0, q1]}

Thực tế, có rất nhiều trạng thái của NFA không có hàm chuyển đến từ trạng thái bắt đầu [q0]. Do đó, thông thường, cách tốt nhất là ta nên xây dựng DFA tương đương bắt đầu từ trạng thái [q0] và thêm vào các trạng thái mới cho DFA chỉ khi có các hàm chuyển từ một trạng thái đã được thêm vào trước đó.

Câu hỏi :

?

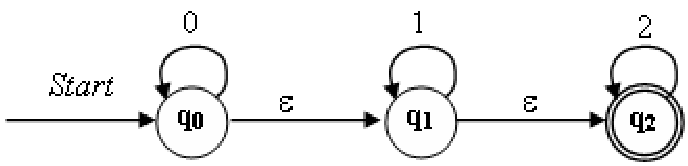
Bạn có nhận xét gì về kích thước giữa một DFA và một NFA tương đương với nó

chấp nhận cùng một tập ngôn ngữ ?

NFA với -dịch chuyển (NFA)

Ta mở rộng mô hình NFA cho phép các phép chuyển trên nhãn rỗng . Sơ đồ chuyển sau đây của một NFA chấp nhận chuỗi gồm một số bất kỳ (có thể là 0) chữ số 0 sau đó là một số bất kỳ chữ số 1 và sau nữa là một số bất kỳ chữ số 2. Thông thường, ta nói NFA chấp nhận một chuỗi w nếu có đường truyền nhãn w từ trạng thái bắt đầu đến một trạng thái kết thúc. Chẳng hạn, chuỗi 002 được chấp nhận bởi đường truyền q0, q0, q0, q1, q2, q2 với các cung nhãn 0, 0, , , 2.

Thí dụ 3.6 : Sơ đồ chuyển của một NFA với -dịch chuyển :



Hình 3.4 - NFA với -dịch chuyển

Định nghĩa: Một cách hình thức ta định nghĩa NFA với -dịch chuyển là bộ 5 thành phần (Q, , , q0, F) với tất cả các thành phần có ý nghĩa như trên, nhưng hàm chuyển là ánh xạ từ Q ({ }) 2Q.

Khái niệm (q, a) gồm tất cả các trạng thái p sao cho có phép chuyển nhãn a từ q tới p, trong đó a là một ký hiệu thuộc hoặc là .

Hàm chuyển trạng thái mở rộng: Ta mở rộng hàm chuyển thành hàm chuyển * ánh xạ từ Q * 2Q. *(q,w) chứa tất cả các trạng thái p sao cho có thể đi từ q tới p theo đường đi nhãn w (có thể chứa cạnh nhãn).

Ta sử dụng -CLOSURE(q) để xác định tập tất cả các đỉnh p sao cho có đường đi từ q tới p với nhãn .

Thí dụ 3.7 : Trong hình 3.4, -CLOSURE(q0) = {q0, q1, q2}.

Vì đường đi chỉ có một đỉnh q0 (không có cung trên đường đi) là đường đi từ q0 tới q0 có tất cả các cạnh nhãn là . Đường đi q0, q1 chỉ ra rằng q1 thuộc -CLOSURE(q0). Và đường đi q0, q1, q2 chỉ ra rằng q2 thuộc -CLOSURE(q0).

Đặt -CLOSURE(P) = q P -CLOSURE(q), trong đó P là một tập các trạng thái và q là một trạng thái. Ta định nghĩa hàm * như sau:

1. *(q, e) = e-CLOSURE(q)

2. *(q, wa) = e-CLOSURE(P),

trong đó tập P = {p có r trong *(q, w) sao cho p (r, a)}, w * và a

Hay *(q, wa) = e-CLOSURE(d(*(q, w), a)

Ta mở rộng   và * trên tập hợp các trạng thái R như sau :

3.   (R, a) = È q R   (q, a), và

4. *(R, w) = Èq R   *(q, w)

Câu hỏi :

?

Hãy so sánh sự khác biệt giữa hàm chuyển   và * ?

Nhận xét : *(q, a) và (q, a) không nhất thiết bằng nhau vì *(q, a) gồm tất cả các trạng thái có thể chuyển đến được từ q trên nhãn a gồm cả đường đi nhãn e, trong khi đó d(q, a) chỉ gồm các trạng thái có thể đến được từ q chỉ bằng các cung nhãn a. Tương tự *(q, e) có thể cũng không bằng (q, e). Vì vậy ta phải phân biệt ký hiệu   và * đối với NFA với e-dịch chuyển.

Ngôn ngữ được chấp nhận bởi NFA :

Ta định nghĩa L(M), ngôn ngữ được chấp nhận bởi NFA M = (Q, , , q0, F) là tập hợp các chuỗi :

L(M) = {w   *(q0, w) có chứa ít nhất một trạng thái trong F}

Thí dụ 3.8 : Xét sơ đồ chuyển của hình 3.4.

Theo khái niệm hình thức, ta có NFA M ({q0, q1, q2}, {0, 1, 2}, , q0, {q2}) với hàm chuyển   như sau :

d	Inputs			

Trạng thái	0	1	2	e
q0	{q0 }	Æ	Æ	{ q1}
q1	Æ	{q1}	Æ	{q1}
q2	Æ	Æ	{q2}	Æ

Xét chuỗi nhập w = 012.

Ta cần tính *(q0, 012)

Ta có : *(q0, e) = e-CLOSURE(q0) = {q0, q1, q2}

vậy *(q0, 0) = e-CLOSURE((*(q0, e), 0)

= e-CLOSURE(d({q0, q1, q2}, 0))

= e-CLOSURE(d(q0, 0) È d(q1, 0) È d(q2, 0))

= e-CLOSURE({q0} È Æ È Æ)

= e-CLOSURE({q0}) = {q0, q1, q2}

và *(q0, 01) = e-CLOSURE((*(q0, 0), 1))

= e-CLOSURE(d({q0, q1, q2}, 1))

= e-CLOSURE(d(q0, 1) È d(q1, 1) È d(q2, 1))

= e-CLOSURE(Æ È {q1} ÈÆ)

= e-CLOSURE({q1}) = {q1, q2}

 *(q0, 012) = e-CLOSURE(d(*(q0, 01), 2))

= e-CLOSURE(d({q1, q2}, 2))

= e-CLOSURE(d(q1, 2) È d(q2, 2))

= e-CLOSURE(Æ È {q2})

= e-CLOSURE({q2}) = {q2}

Do   *(q0, 012) có chứa trạng thái q2   F nên chuỗi w   L(M).

Giải thuật mô phỏng hoạt động của một NFA   :

. Input : Chuỗi nhập x được kết thúc bởi $.

. Output : Câu trả lời "YES" nếu NFA chấp nhận chuỗi x và "NO" nếu ngược lại.

. Giải thuật :

q := e-CLOSURE(q0);

c := nextchar ; { c là ký hiệu nhập được đọc tiếp theo }

While c <> $ do

begin

q := e-CLOSURE(d(q, c));

c := nextchar ;

end

If q in F then write ("YES") else write ("NO");

Sự tương đương giữa NFA có và không có e-dịch chuyển

Tương tự như NFA, khả năng có thể thực hiện phép chuyển trên nhãn e của NFAe cũng không làm cho NFAe chấp nhận được các tập hợp không chính quy. Ta có thể dẫn chứng điều này bằng cách mô phỏng hoạt động của một NFAe bởi một NFA không có e-dịch chuyển.

ĐỊNH LÝ 3.2 : Nếu L được chấp nhận bởi một NFA có e-dịch chuyển thì L cũng được chấp nhận bởi một NFA không có e-dịch chuyển.

Chứng minh

Đặt M (Q, , , q0, F) là NFA với e-dịch chuyển.

Ta xây dựng NFA M'(Q, , ', q0, F') tương đương không có e-dịch chuyển, trong đó:

F {q0} nếu e-CLOSURE(q0) chứa một trạng thái thuộc F

. F' =

F trong các trường hợp còn lại

. '(q, a) là *(q, a) với q Q và a . Chú ý rằng M' không có e-dịch chuyển nên ta có thể dùng ' thay cho *', nhưng phải phân biệt và *.

Ta chứng minh bằng quy nạp trên x rằng '(q0, x) = *(q0, x). Tuy nhiên, điều đó có thể không đúng với x = e vì '(q0, e) = {q0} trong khi *(q0, e) = e-CLOSURE(q0). Do đó, cơ sở quy nạp bắt đầu với độ dài chuỗi là 1.

Với | x | = 1 thì x là một ký hiệu a và '(q, a) = *(q, a) theo định nghĩa '.

Xét | x | > 1: đặt x = wa với a là một ký hiệu trong .

Ta có '(q, wa) = '('(q0, w), a)

Theo giả thiết quy nạp thì '(q0, w) = *(q0, w). Đặt *(q0, w) = P, ta cần chỉ ra rằng (P, a) = *(q0, wa).

Ta có ’(P, a) = Èq P ’(q, a) = Èq P *(q, a).

Hơn nữa vì P = *(q0, w) nên Èq P *(q, a) = *(q0, wa) (theo quy tắc 2 trong định nghĩa *).

Vậy ’(q0, wa) = *(q0, wa)

Để đầy đủ chứng minh ta còn phải chỉ ra rằng ’(q0, x) chứa một trạng thái trong F’ nếu và chỉ nếu *(q0, x) chứa một trạng thái trong F.

Nếu x = e thì điều đó hiển nhiên đúng (theo định nghĩa của F’)

Nếu x e thì ta đặt x = wa với a .

Nếu *(q0, x) chứa một trạng thái trong F thì chắc chắn ’(q0, x) chứa cùng trạng thái trong F’. Ngược lại, nếu ’(q0, x) chứa một trạng thái trong F’ khác hơn q0 thì (q0, x) phải chứa một trạng thái trong F (vì tập F và F’ chỉ chênh lệch nhau trạng thái q0). Nếu ’(q0, x) có chứa trạng thái q0 và q0 cũng là một trạng thái thuộc tập trạng thái kết thúc F thì vì *(q0, x) = e-CLOSURE((*(q0, w),a)), nên trạng thái chung trong e-CLOSURE(q0) và trong F phải ở trong *(q0, x).

Thí dụ 3.9 : Chuyển NFA với e-dịch chuyển ở hình 3.4 sang dạng NFA không có chứa e-dịch chuyển.

Ta xây dựng NFA tương đương M’(Q, , ’, q0, F’) chấp nhận L(M) với các thành phần : . Q = {q0, q1, q2}

. å = {0, 1, 2}

. Trạng thái bắt đầu : q0

. F' = {q0, q2} do e-CLOSURE(q0) = {q0, q1, q2} có chứa q2 F

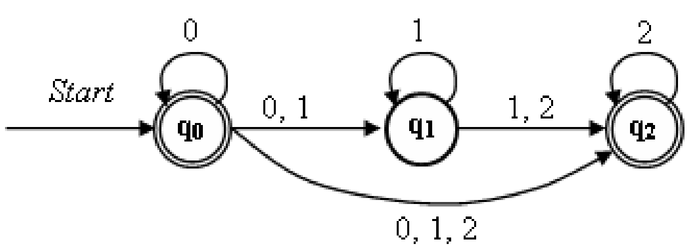
. Hàm chuyển ’ của M’ được xác định theo công thức :

d’(q, a) = d*(q, a) = e-CLOSURE(d(d*(q0, e), a)

Kết quả được chỉ ra trong bảng hàm chuyển sau :

d’	Inputs		
Trạng thái	0	1	2
q0	{q0, q1, q2}	{q1, q2}	{q2}
q1	Æ	{q1, q2}	{q2}
q2	Æ	Æ	{q2}

Sơ đồ chuyển trạng thái:



Hình 3.5 - NFA tương đương cho thí dụ 3.9

Giải thuật xây dựng DFA từ NFA

Qua khảo sát các dạng mở rộng từ mô hình ôtômát hữu hạn ban đầu, ta thấy DFA thực chất là một trường hợp đặc biệt của NFA, nhưng :

- Nó không có sự truyền rỗng (truyền trên nhãn e)

- Với mỗi trạng thái q và ký hiệu nhập a, chỉ có duy nhất một đường truyền đến một trạng thái khác.

Giả sử mỗi trạng thái của DFA là một tập trạng thái của NFA, DFA dùng trạng thái của mình để lưu giữ tất cả các trạng thái của NFA đạt được sau khi NFA đọc một ký tự nhập. Như vậy sau khi đọc các ký tự nhập a1, a2, ... , an, DFA ở trạng thái là tập con của các trạng thái thuộc NFA, đạt được khi NFA đi từ trạng thái bắt đầu theo một con đường nào đó có tên a1a2 ... an. Số trạng thái của DFA lúc đó phải bằng số phần tử trong tập lũy thừa của số trạng thái NFA. Song, trên thực tế trường hợp xấu nhất này ít khi xảy ra. Các trạng thái thực sự được dùng trong sơ đồ chuyển cho một DFA sẽ được xác định theo các phép chuyển trạng thái trên nhãn là mọi ký hiệu từ trạng thái bắt đẩu của DFA, và sau đó lần lượt được bổ sung thêm vào tập trạng thái nếu như nó chưa có trong đó.

Giải thuật chi tiết được trình bày như sau :

Input: Một ôtômát hữu hạn không đơn định NFA.

Output: Một ôtômát hữu hạn đơn định DFA nhận dạng cùng ngôn ngữ như NFA.

Phương pháp: Xây dựng bảng hàm chuyển cho DFA mô phỏng đồng thời tất cả các chuyển dịch của NFA trên chuỗi nhập cho trước.

Ta dùng các tác vụ sau để lưu giữ các tập trạng thái của NFA :

(q : là một trạng thái của NFA, T : là tập trạng thái của NFA)

a) e-closure(q) : là tập trạng thái của NFA đạt được từ trạng thái q trên sự truyền rỗng.

b) e-closure(T) : là tập trạng thái của NFA đạt được từ tất cả các trạng thái q thuộc tập T trên sự truyền rỗng.

c) d(T, a) : là tập trạng thái của NFA đạt được từ tất cả các trạng thái q thuộc tập T trên sự truyền bằng ký hiệu a.

Phân tích:

Trước khi đọc vào một ký tự nhập, DFA có thể ở một trạng thái bất kỳ trong các trạng thái thuộc e-closure(q0) với q0 là trạng thái bắt đầu của NFA. Gọi trạng thái này là T. Giả sử các trạng thái của T là các trạng thái đạt được từ q0 trên các ký hiệu nhập và giả sử a là ký hiệu nhập kế tiếp. Khi đọc a, NFA có thể chuyển đến một trạng thái bất kỳ trong tập trạng thái d(T, a). Khi chúng ta cho phép sự truyền rỗng, NFA có thể ở bất kỳ trạng thái nào trong e-closure(d(T, a)) sau khi đã đọc a.

Giải thuật :

Trạng thái bắt đầu -closure(q0) chỉ là một trạng thái trong các trạng thái

của DFA và trạng thái này chưa được đánh dấu;

While Có một trạng thái T của DFA chưa được đánh dấu do

Begin

Đánh dấu T; { xét trạng thái T}

For Với mỗi ký hiệu nhập a do

begin

U:= -closure((T, a))

If U không có trong tập trạng thái của DFA then

begin

Thêm U vào tập các trạng thái của DFA và trạng thái

này chưa được đánh dấu;

 [T, a] := U; { [T, a] là phần tử của bảng chuyển DFA}end;

end;

End;

Ta xây dựng các trạng thái và bảng hàm chuyển cho DFA theo cách như sau :

- Mỗi trạng thái của DFA tượng trưng bởi một tập trạng thái của NFA mà NFA có thể chuyển đến sau khi đọc một chuỗi ký hiệu nhập gồm: tất cả sự truyền rỗng có thể xảy ra trước hoặc sau các ký hiệu được đọc.

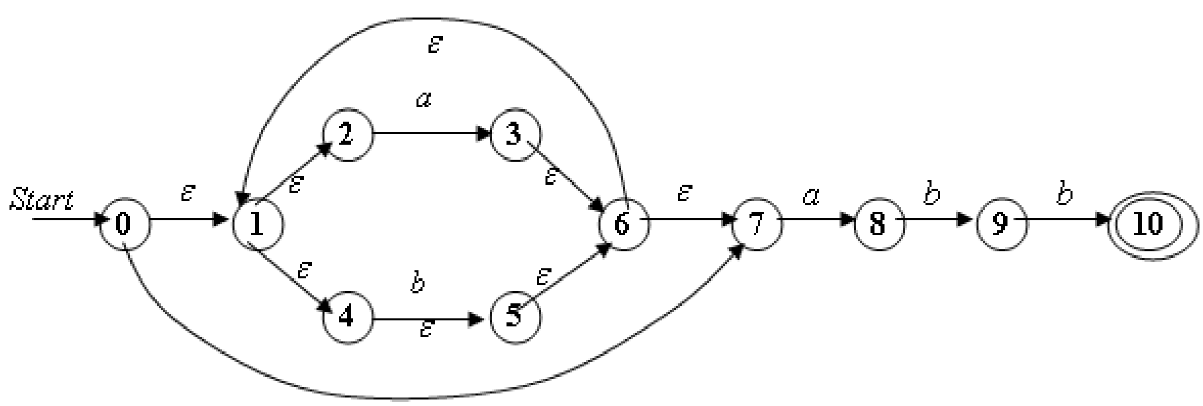
- Trạng thái bắt đầu của DFA là e-closure(q0)

- Các trạng thái và hàm chuyển sẽ được thêm vào D bằng giải thuật trên.

- Một trạng thái của DFA là trạng thái kết thúc nếu nó là tập các trạng thái của NFA chứa ít nhất một trạng thái kết thúc của NFA.

Việc tính toán e-closure(T) có thể xem như quá trình tìm kiếm một đồ thị của các nút từ các nút cho trước và đồ thị bao gồm toàn những cạnh có nhãn e của NFA. Giải thuật đơn giản để tìm e-closure(T) là dùng Stack để lưu giữ các trạng thái mà cạnh của chúng chưa được kiểm tra cho sự truyền rỗng.

Thí dụ 3.10 : Tạo DFA từ NFAe sau



Hình 3.6 – Thí dụ chuyển NFA có -dịch chuyển

Các bước xây dựng tập trạng thái cho DFA :

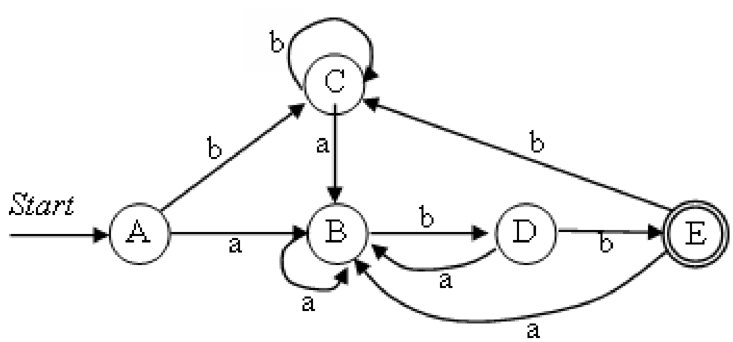
1. Trạng thái bắt đầu của DFA : -closure(0) = {0, 1, 2, 4, 7} = A*
2. -closure((A, a)) = -closure({3, 8}) = {1, 2, 3, 4, 6, 7, 8} = B*
3. -closure((A, b)) = -closure({5}) = {1, 2, 4, 5, 6, 7} = C*
4. -closure((B, a)) = -closure({3, 8}) = B
5. -closure((B, b)) = -closure({5, 9}) = {1, 2, 4, 5, 6, 7, 9} = D*
6. -closure((C, a)) = -closure({3, 8}) = B
7. -closure((C, b)) = -closure({5}) = C
8. -closure((D, a)) = -closure({3, 8}) = B
9. -closure((D, b)) = -closure({5, 10}) = {1, 2, 4, 5, 6, 7, 10} = E*
10. -closure((E, a)) = -closure({3, 8}) = B
11. -closure((E, b)) = -closure({5}) = C

Từ các tập trạng thái này, ta xác định được A là trạng thái bắt đầu, E là trạng thái kết thúc (vì trong E có chứa trạng thái 10 là trạng thái kết thúc của NFA) và bảng hàm chuyển của DFA như sau :

Trạng thái	Ký hiệu nhập	
	a	b
A	B	C

B	B	D
C	B	C
D	B	E
E	B	C

Từ bảng hàm chuyển như trên, ta xây dựng sơ đồ chuyển trạng thái cho DFA tương đương nhận dạng cùng ngôn ngữ có dạng như sau :



Hình 3.7 – DFA tương đương cho thí dụ 3.10

Nhận xét : Mặc dù có sự khác nhau trong định nghĩa, ta thấy dạng không đơn định NFA được định nghĩa tổng quát hơn dạng đơn định DFA, nhưng rõ ràng khả năng nhận dạng cùng lớp ngôn ngữ của chúng là tương đương nhau. Trong thực tế, các máy tính số hoàn toàn là đơn định, trạng thái của chúng tại mỗi thời điểm là xác định được duy nhất từ một chuỗi nhập bất kỳ và trạng thái bắt đầu.

Câu hỏi :

?

Tại sao cần định nghĩa dạng không đơn định ?

Một số gợi ý câu trả lời:

1. Trong một số các bài toán mang tính chọn lựa, có nhiều hướng giải quyết (nhiều cách đi) như trong các chương trình trò chơi (games) thì thông thường hướng giải quyết tốt nhất (cách đi tốt nhất) là không biết trước được, nhưng có thể tìm thấy được bằng cách sử dụng chiến lược tìm kiếm quay lui (back-tracking). Khi có một vài khả năng chọn lựa có thể, ta chọn một khả năng trong chúng và đi theo hướng đó cho đến khi xác định hướng đó là tốt nhất hay chưa. Nếu chưa phải là hướng tốt nhất, ta phải quay về điểm quyết định cuối cùng trước đó và thử khảo sát theo một hướng khác. Một giải thuật mô phỏng quá trình tìm kiếm quay lui này là một giải thuật không đơn định.
2. Không đơn định đôi khi còn rất hữu hiệu trong việc giúp giải quyết các bài toán dễ dàng. Chẳng hạn, trong một số bài toán thì việc xây dựng một NFA có vẻ tự nhiên và đơn giản hơn việc tìm một DFA cho chúng. Tương tự như vậy, không đơn định còn là một cơ chế hiệu quả dùng mô tả văn phạm sinh ra ngôn ngữ một cách súc tích (sự chọn lựa các luật sinh sinh từ cùng một biến).
3. Trong thực tế, một vài kết quả là dễ dàng được chứng minh đối với NFA hơn là DFA. Vì vậy việc cho phép cơ chế không đơn định thường làm đơn giản hóa các lý luận hình thức mà không ảnh hưởng đến tính tổng quát của kết luận.

BIỂU THỨC CHÍNH QUY (RE : Regular Expressions)

Lớp ngôn ngữ được chấp nhận bởi một ôtômát hữu hạn cũng có thể được mô tả thông qua một dạng biểu thức ngắn gọn và súc tích gọi là biểu thức chính quy. Trong phần này, chúng ta sẽ giới thiệu sự kết hợp của các phép toán hợp, nối kết và bao đóng Kleene trên các tập hợp chuỗi để định nghĩa biểu thức chính quy và chứng tỏ rằng lớp ngôn ngữ được chấp nhận bởi một ôtômát hữu hạn thì thực sự là lớp ngôn ngữ được mô tả bởi biểu thức chính quy.

Định nghĩa

Cho   là một bộ chữ cái. Biểu thức chính quy trên   và các tập hợp mà chúng mô tả được định nghĩa một cách đệ quy như sau:

1)   là biểu thức chính quy ký hiệu cho tập rỗng

2)   là biểu thức chính quy ký hiệu cho tập { }

3) a   , a là biểu thức chính quy ký hiệu cho tập {a}

4) Nếu r và s là các biểu thức chính quy ký hiệu cho các tập hợp R và S thì (r + s), (rs) và (r*) là các biểu thức chính quy ký hiệu cho các tập hợp R   S, RS, R* tương ứng.

Trong khi viết biểu thức chính quy ta có thể bỏ bớt các dấu ngoặc đơn với lưu ý rằng thứ tự ưu tiên của các phép toán xếp theo thứ tự giảm dần là: phép bao đóng, phép nối kết, phép hợp.

Chẳng hạn : Biểu thức ((0(1*)) + 1) có thể viết là 01*+ 1.

Câu hỏi :

?

Như trên ta nói, biểu thức chính quy dùng ký hiệu cho một lớp ngôn ngữ. Bạn

hãy thử liệt kê một vài chuỗi và hình dung lớp ngôn ngữ được ký hiệu bởi biểu

thức chính quy r = 01*+ 1 trên ?

Phép toán bao đóng dương cũng có thể được sử dụng khi viết biểu thức chính quy. Ta có thể viết rút gọn rr* hay r*r thành r+.

Nếu cần thiết phân biệt thì ta sẽ dùng ký hiệu r cho biểu thức chính quy r và L(r) cho ngôn ngữ được ký hiệu bởi biểu thức chính quy r; ngược lại

một cách tổng quát, ta có thể dùng r cho cả hai.

Thí dụ 3.11 : Một số biểu thức chính quy ký hiệu cho các ngôn ngữ :

. 00 là biểu thức chính quy biểu diễn tập {00}.

. (0+1)* ký hiệu cho tập hợp tất cả các chuỗi số 0 và số 1, kể cả chuỗi rỗng

= {e, 0, 1, 00, 01, 10, 11, 010, 011, 0010 ... }

. (0+1)*00(0+1)* ký hiệu cho tập hợp tất cả các chuỗi 0,1 có ít nhất hai số 0 liên tiếp.

= {00, 000, 100, 0000, 0001, 1000, 1001, 011001, ... }

. (1+10)* ký hiệu cho tất cả các chuỗi 0, 1 bắt đầu bằng số 1 và không có hai số 0 liên tiếp = {e, 1, 10, 11, 1010, 111, 101010, ... }

. (0+)(1+10)* ký hiệu cho tất cả các chuỗi không có hai số 0 liên tiếp.

= {e, 0, 01, 010, 1, 10, 01010, 0111, ... }

. (0+1)*011 ký hiệu cho tất cả các chuỗi 0, 1 tận cùng bởi 011.

= {011, 0011, 1011, 00011, 11011, ... }

. 0*1*2* ký hiệu cho tất cả các chuỗi có một số bất kỳ các số 0, theo sau là một số bất kỳ số 1 và sau nữa là một số bất kỳ số 2.

= {e, 0, 1, 2, 01, 02, 12, 012, 0012, 0112, ... }

. 00*11*22* ký hiệu cho tất cả các chuỗi trong tập 0*1*2* với ít nhất một trong mỗi ký hiệu. 00*11*22* có thể được viết gọn thành 0+1+2+

Thí dụ 3.12 : Biểu thức chính quy ký hiệu cho tập hợp các chuỗi tên biến đúng trong ngôn ngữ lập trình Pascal :

Một chuỗi tên biến (identifiers) được gọi là hợp lệ trong một chương trình Pascal nếu như nó bắt đầu bằng ít nhất một chữ cái và theo sau đó là các chữ cái, số, ký hiệu underline hoặc một vài ký hiệu cho phép khác trên bàn phím máy tính.

Biểu thức chính quy có dạng như sau :

r = (A + …+ Z + a + … + z) (A + …+ Z + a + … + z + 0 + … + 9 + _ + …)*

Thí dụ 3.13 : Biểu thức chính quy ký hiệu cho tập hợp các số nguyên trong ngôn ngữ lập trình Pascal :

Một chuỗi số nguyên trong một chương trình Pascal có thể bắt đầu bằng dấu âm (-) hoặc dấu dương (+) hay không chứa ký hiệu dấu, và theo sau đó là một chuỗi các ký hiệu số với ít nhất là một số.

Biểu thức chính quy có dạng như sau :

r = (‘+’ + ‘-‘ +) (0 + … + 9 (0 + … +9)*

Nhận xét : Thông thường, việc tìm một biểu thức chính quy ký hiệu cho một ngôn ngữ khó hơn việc xác định ngôn ngữ được ký hiệu bởi một biểu thức chính quy vì không có giải thuật cho loại bài toán này.

Một số tính chất đại số của biểu thức chính quy

Dễ dàng chứng minh rằng, nếu cho r, s, t là các biểu thức chính quy thì ta có các đẳng thức sau :

1.r + s = s + r2. r + r = r

3.r + (s+t) = (r+s) + t4. r (st) = (rs) t

5. r (s+t) = rs + rt6. (r+s) t = rt + st

7. r = r = r8. r = r =

9. r + = r10. * =

11. (+ r)* = r*12.r + r* = r*

13. (r*)* = r*14. (r* s*)* = (r+s)*

Trong đó, ta có r = s có nghĩa là L(r) = L(s).

SỰ TƯƠNG ĐƯƠNG GIỮA ÔTÔMÁT HỮU HẠN VÀ BIỂU THỨC CHÍNH QUY

Như trên đã nói, các ngôn ngữ được chấp nhận bởi ôtômát hữu hạn cũng là các ngôn ngữ được mô tả bởi biểu thức chính quy. Chính vì sự tương đương này, mà người ta gọi ngôn ngữ chấp nhận bởi ôtômát hữu hạn là các tập chính quy. Trong phần này, thông qua hai định lý, ta sẽ chỉ ra bằng quy nạp theo kích thước của (số phép toán trong) biểu thức chính quy rằng có tồn tại một NFA với e-dịch chuyển chấp nhận cùng ngôn ngữ; đồng thời với mỗi DFA cũng có một biểu thức chính quy xác định chính ngôn ngữ của nó.

ĐỊNH LÝ 3.3: Nếu r là biểu thức chính quy thì tồn tại một NFA với e-dịch chuyển chấp nhận L(r).

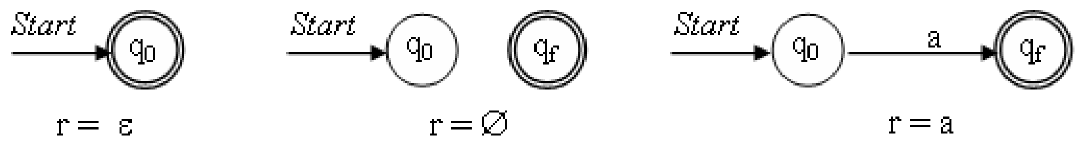
Chứng minh

Ta sẽ chứng minh quy nạp theo số phép toán của biểu thức chính quy r rằng có tồn tại một NFA M với e-dịch chuyển có một trạng thái kết thúc và không có các phép chuyển khỏi trạng thái này chấp nhận biểu thức chính quy r: L(M) = L(r).

. r không có phép toán:

Vậy r phải là , e hoặc a (với a).

Các NFA dưới đây thoả mãn điều kiện:



Hình 3.7 - Các NFAe cho các kết hợp đơn

. r có chứa các phép toán:

Giả sử định lý đúng với r có ít hơn i phép toán, i   1.

Xét r có i phép toán. Có 3 trường hợp :

1) r = r1+ r2.

Cả hai biểu thức chính quy r1, r2 có ít hơn i phép toán, vậy ta có 2 ôtômát hữu hạn NFA M1 (Q1, 1, 1, q1, {f1}) và M2 (Q2, 2, 2, q2, {f2}) sao cho L(M1) = L(r1) và L(M2) = L(r2). Vì các trạng thái có thể thay đổi tên nên ta giả sử hai tập trạng thái Q1 và Q2 là rời nhau. Đặt q0 là trạng thái bắt đầu mới và {f0} là tập trạng thái kết thúc mới, ta xây dựng NFA M (Q1 Q2 {q0, f0}, 1   2, , q0, {f0}), trong đó được xác định như sau:

. d(q0, e) = {q1, q2}

. d(q, a) = d1(q, a) với q Q1 - {f1} và a   1 { }

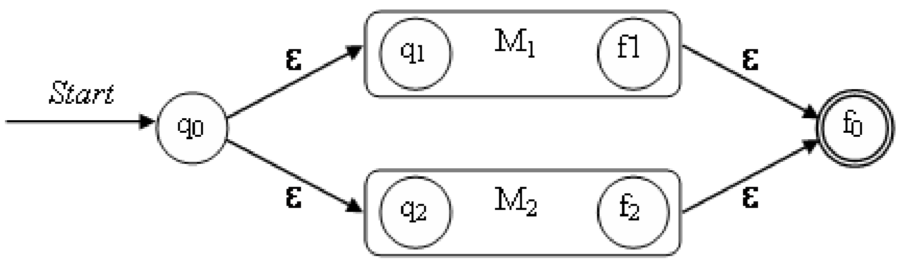
. d(q, a) = d2(q, a) với q Q2 - {f2} và a   2 { }

. d(f1, e) = d(f2, e) = {f0}

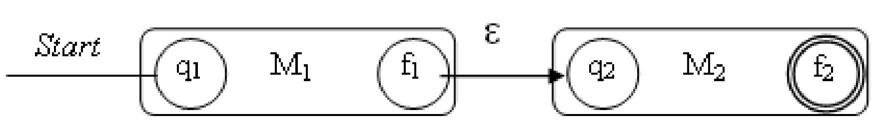
Chú ý do giả thiết quy nạp là không có phép chuyển nào ra khỏi f1, f2 trong M1, M2. Vì vậy tất cả các phép chuyển của M1 và M2 đều có trong M. Cách xây dựng M chỉ ra trong hình a. Bất kỳ đường đi nào trong sơ đồ chuyển của M từ q0 tới f0 phải bắt đầu bằng cách đi tới q1 hoặc q2 bằng nhãn . Nếu đường đi qua q1 thì nó theo một đường đi nào đó trong M1 tới f1 rồi sau đó tới f0 bằng nhãn .

Tương tự trong trường hợp đường đi qua q2. Có một đường đi từ q0 đến f0 nhãn x khi và chỉ khi có đường đi nhãn x trong M1 từ q1 đến f1 hoặc có đường đi nhãn x trong M2 từ q2 đến f2.

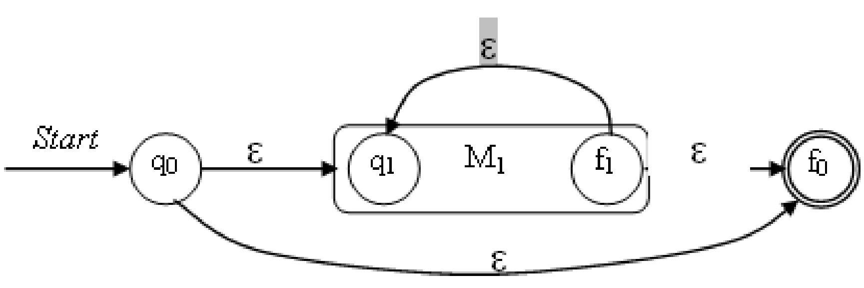
Vậy L(M) = L(M1)   L(M2)



Hình a - Phép hợp



Hình b - Phép nối kết



Hình c - Phép bao đóng

Hình 3.8 - Các NFAe cho kết hợp phức

2) r = r1 r2

Đặt M1 và M2 là các ôtômát NFA như trong trường hợp trên và ta xây dựng ôtômát M (Q, , , {q1}, {f2}), trong đó được xác định như sau:

. (q, a) = 1(q, a) với q Q1 - {f1} và a 1 { }

. d(f1, e) = {q2}

. (q, a) = 2(q, a) với q Q2 và a 2 { }

Cách xây dựng M chỉ ra trong hình b. Mỗi đường đi trong M từ q1 tới f2 là đường đi có nhãn x từ q1 tới f1 sau đó là một cung từ f1 tới q2 nhãn và tiếp đến là đường đi từ q2 tới f2.

Vậy L(M) = {xy x L(M1) và y L(M2)} hay L(M) = L(M1) L(M2).

3) r = r*

Đặt M1 (Q1, 1, 1, q1, {f1}) và L(M1) = r1.

Xây dựng M (Q1 {q0, f0} 1, , q0, {f0}), trong đó được cho:

. d(q0, e) = d(f1, e) = {q1, f0}

. d(q, a) = d1(q, a) với q Q1 - {f1} và a 1 { }

Cách xây dựng M được chỉ ra trong hình c. Mỗi đường đi từ q0 tới f0 gồm: hoặc đường đi từ q0 tới f0 bằng nhãn ; hoặc đường đi từ q0 tới q1 bằng nhãn và sau đó là đường đi từ q1 tới f1 trên chuỗi thuộc L(M), rồi đến f0 bằng nhãn . Như vậy có đường đi từ q0 tới f0 nhãn là x nếu và chỉ nếu ta có thể viết x = x1 x2 ... xj với j 0 (trường hợp j = 0 khi x =) xi L(M1). Vậy L(M) = L(M1)*.

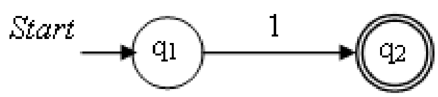
Thí dụ 3.14 : Xây dựng NFA chấp nhận lớp ngôn ngữ được ký hiệu bởi biểu thức chính quy r = 01* + 1.

Ta thấy L(r) = { 1, 0, 01, 011, 0111, 01111, 011111, … } là tập ngôn ngữ chứa các bit đơn 0, 1 và các chuỗi bit nhị phân bắt đầu bằng bit 0, theo sau là một chuỗi bit 1 với độ dài tuỳ ý.

Theo quy luật thứ tự ưu tiên, biểu thức 01* + 1 thực chất là (0(1*)) + 1, vì vậy nó có dạng r1 + r2 với r1 = 01* và r2 = 1.

Ta sẽ lần lượt xây dựng các NFA cho các biểu thức chính quy con, sau đó dựa vào các quy tắc kết hợp để xây dựng NFA cho toàn bộ biểu thức chính quy đã cho.

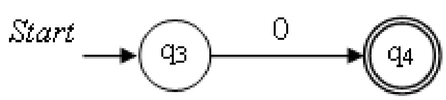
. NFA cho r2 = 1 dễ dàng để xây dựng :



. NFA cho r1 = 01*:

Ta tách r1 = r3 r4 , trong đó r3 = 0 và r4 = 1*

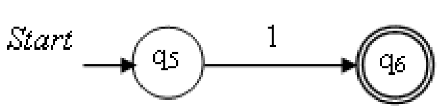
+ NFA cho r3 = 0 :



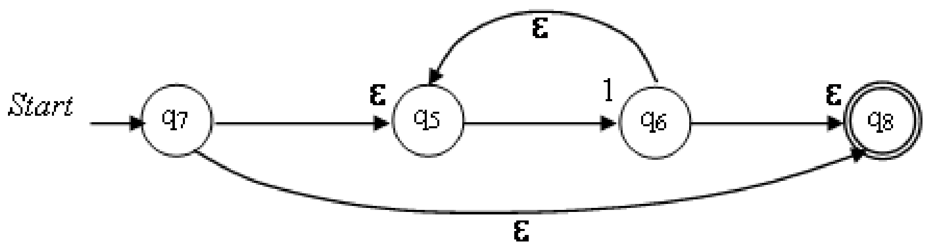
+ NFA r4 = 1* :

Ta viết r4 = r5*, trong đó r5 = 1.

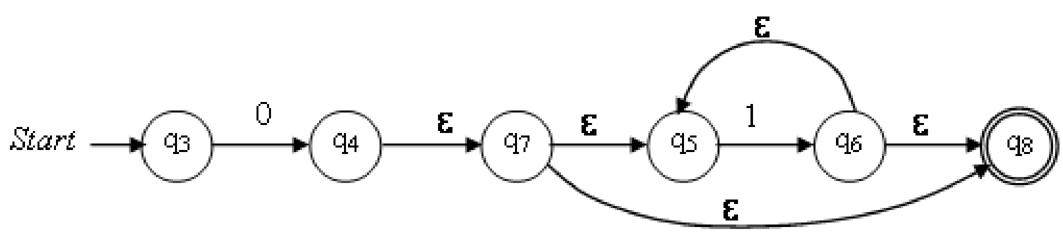
NFA cho r5 = 1 :



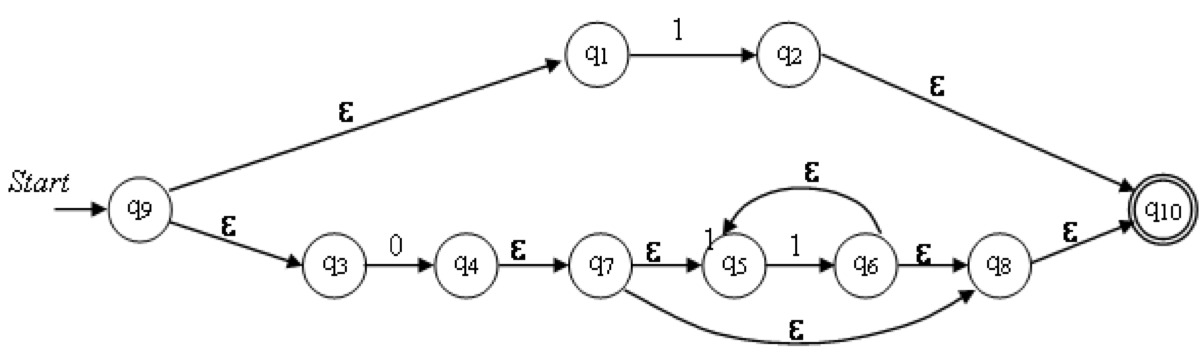
Theo quy tắc 3) ta xây dựng được NFA cho r4 = r5* = 1* như sau :



Theo quy tắc 2) ta xây dựng được NFA cho r1 = r3 r4 = 01* như sau :



Cuối cùng, theo quy tắc 1) ta xây dựng NFA cho r = r1 + r2 = 01*+ 1 như sau :



Hình 3.9 - NFAe cho ví dụ 3.13

Phần chứng minh của Định lý 3.3 trên cũng chính là cơ sở của giải thuật chuyển đổi một biểu thức chính quy thành ôtômát hữu hạn. Một điểm cần lưu ý là thứ tự ưu tiên của các phép toán được sử dụng trong biểu thức chính quy, điều này rất quan trọng cho quá trình tách biểu thức chính quy thành các biểu thức con trong những trường hợp viết biểu thức chính quy ở dạng tắt (không có dấu ngoặc).

Bây giờ, ta cần chứng tỏ rằng mọi tập hợp được chấp nhận bởi một ôtômát hữu hạn thì cũng được ký hiệu bởi một số biểu thức chính quy.

ĐỊNH LÝ 3.4 : Nếu L được chấp nhận bởi một DFA, thì L được ký hiệu bởi một biểu thức chính quy.

Chứng minh

Đặt L là tập hợp được chấp nhận bởi DFA M ({q1, q2,..., qn}, , , q1, F).

Đặt Rkij là tập hợp tất cả các chuỗi x sao cho (qi, x) = qj và nếu (qi, y) = ql, với y là tiền tố bất kỳ của x, khác x hoặc , thì l k. Tức là Rkij là tập hợp tất cả các chuỗi làm cho ôtômát đi từ trạng thái qi tới qj không đi ngang qua trạng thái nào (được đánh số) lớn hơn k. (Chú ý, khái niệm "đi ngang qua một trạng thái" có nghĩa là có phép chuyển vào và ra khỏi trạng thái đó, nên i hoặc j đều có thể lớn hơn k). Vì chỉ có n trạng thái nên Rnij sẽ là tập hợp tất cả các chuỗi làm ôtômát đi từ qi tới qj.

Ta định nghĩa Rkij một cách đệ quy như sau:

Rkij = Rk-1ik (Rk-1kk)* Rk-1kj È Rk-1ij (1)

{ a (qi, a) = qj } nếu i j

R0ij =

{ a (qi, a) = qj } { } nếu i = j

Một cách hình thức, Rkij định nghĩa như trên là các chuỗi nhập hay nguyên nhân đưa M từ qi tới qj không đi ngang qua trạng thái cao hơn qk, nghĩa là xảy ra hoặc một trong hai trường hợp sau :

1. Nằm trong Rk-1ij (để không bao giờ đi ngang qua một trạng thái nào cao như qk).
2. Bao gồm một chuỗi trong Rk-1ik (chuỗi làm M chuyển đến qk), theo sau bởi không hoặc nhiều chuỗi trong Rk-1kk (chuỗi làm M chuyển từ qk trở về qk mà không ngang qua qk hoặc một trạng thái nào cao

hơn) và cuối cùng là một chuỗi trong Rk-1kj (chuỗi làm M chuyển từ qk đến qj).

Ta sẽ chỉ ra rằng với mỗi i, j và k tồn tại biểu thức chính quy rkij ký hiệu cho ngôn ngữ Rkij. Ta quy nạp theo k như sau:

. k = 0 : khi đó R0ij là tập hợp hữu hạn các chuỗi có một ký hiệu hoặc . Vậy r0ij có thể viết dưới dạng a1 + a2 + ... + ap (hoặc a1 + a2 + ... + ap+ nếu i = j). Trong đó {a1, a2,..., ap} là tập hợp tất cả các ký hiệu a sao cho (qi, a) = qj. Nếu không có ký hiệu a nào như thế thì (hoặc khi i = j) ký hiệu cho r0ij.

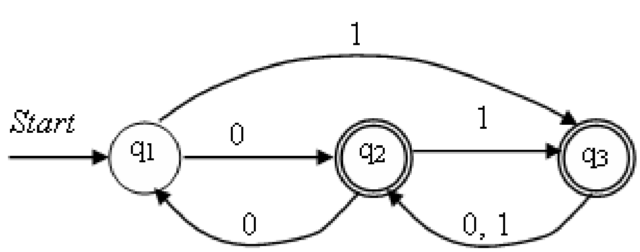
. Công thức (1) cho Rkij chỉ liên quan đến các phép toán trên biểu thức chính quy: hợp, nối kết, và bao đóng. Hơn nữa theo giả thiết quy nạp, với mỗi l, k và m tồn tại biểu thức chính quy rk-1lm sao cho L(rk-1lm) = Rk-1lm. Vậy đối với rkij ta có thể chọn biểu thức chính quy :

(rk-1ik) (rk-1kk)* (rk-1kj) + rk-1ij

Cuối cùng ta có nhận xét rằng L(M) = Èqj F Rn1j vì Rn1j ký hiệu cho tất cả các nhãn của tất cả các đường đi từ q1 tới qj.

Vậy L(M) được ký hiệu bởi biểu thức chính quy r = rn1j1 + rn1j2+ ... + rn1jp, trong đó tập F = {qj1, qj2,..., qjp}

Thí dụ 3.15 : Viết biểu thức chính quy ký hiệu cho ngôn ngữ được chấp nhận bởi DFA sau :



Hình 3.10 – DFA cho ví dụ 3.13

Gọi DFA được chỉ ra trong hình 3.10 là M ({q1, q2, q3}, {0, 1}, , q1, {q2, q3}). Ta thấy, tập hợp tất cả các chuỗi được chấp nhận bởi DFA trên là các chuỗi làm cho ôtômát chuyển từ trạng thái bắt đầu q1 đến một trong hai trạng thái kết thúc q2 và q3 và không chuyển qua số tối đa là 3 (k = 3) trạng thái của ôtômát. Vậy ta cần viết biểu thức chính quy ký hiệu cho tập hợp này như sau :

r = r312 + r313

Theo công thức đã được chứng minh trong Định lý, ta có thể tính được các giá trị rkij với i, j là chỉ số các trạng thái từ 1 đến 3 và với k = 0, 1 và 2, như chỉ ra trong bảng sau:

	k = 0	k = 1	k = 2
rk11			(00)*
rk12	0	0	0(00)*
rk13	1	1	0*1
rk21	0	0	0(00)*
rk22		+ 00	(00)*
rk23	1	1 + 01	0*1
rk31			(0 + 1)(00)*0
rk32	0 + 1	0 + 1	(0 + 1)(00)*
rk33			+ (0 + 1)0*1

Bằng cách dùng các công thức tương đương như (r + s) t = rt + st và (+ r)* = r* để đơn giản các biểu thức, chẳng hạn khi tính biểu thức :

r122 = r021 (r011)* r012 + r022 = 0( )*0 +   = 00 +

Tương tự, khi đơn giản biểu thức

r213 = r112 (r122)* r123 + r113 = 0(00 +)*(1 + 01) + 1

ta nhận thấy (00 +)* tương đương với (00)* và (1 + 01) thì tương đương với (+ 0)1 nên ta rút gọn :

r213 = 0(00)*(+ 0)1 + 1

Mặt khác, chú ý rằng (00)*(+ 0) thì tương đương với 0*, vì thế 0(00)*(+ 0)1 + 1 cũng bằng 00*1 + 1 hay cuối cùng là 0*1.

Để hoàn thành việc xây dựng biểu thức chính quy cho M, ta cần tính r312 và r313. Ta viết:

r312 = r213 (r233)* r232 + r212

= 0*1(+ (0 + 1)0*1)*(0 + 1)(00)* + 0(00)*

= 0*1((0 + 1)0*1)*(0 + 1)(00)* + 0(00)*

và

r313 = r213 (r233)* r233 + r213

= 0*1(+ (0 + 1)0*1)*(+ (0 + 1))0*1) + 0*1

= 0*1((0 + 1)0*1)*

Vậy biểu thức chính quy có dạng :

r = r312 + r313 = 0*1((0 + 1)0*1)*(+ (0 + 1)(00)*) + 0(00)*

MỘT VÀI ỨNG DỤNG CÚA ÔTÔMÁT HỮU HẠN

Có nhiều kiểu phần mềm thiết kế nhằm đặc tả sự chuyển đổi tự động từ dạng biểu thức chính quy sang việc cài đặt máy tính một cách hiệu quả tương ứng với ôtômát hữu hạn. Trong phần này, ta sẽ đề cập đến hai ứng dụng trong số chúng.

Bộ phân tích từ vựng

Các ký hiệu từ vựng (token) trong một ngôn ngữ lập trình thì hầu hết không có sự ngọai lệ, được biểu diễn như các tập hợp chính quy. Chẳng hạn, các định danh của ALGOL: các chữ cái viết hoa hoặc thường, theo sau bởi một dãy bất kỳ của chữ cái (letter) hoặc chữ số (digit) với độ dài không giới hạn có thể được biểu diễn như sau :

(letter) (letter + digit)*

trong đó "letter" thay thế cho A + B +...+ Z + a + b +...+ z và "digit" là 0 + 1 +...+ 9.

Một ví dụ khác, các định danh của FORTRAN có độ dài giới hạn là 6 và các chữ cái chỉ cho phép dùng chữ viết hoa hoặc ký hiệu $ được biểu diễn như sau :

(letter) (+ letter + digit)5

với "letter" là $ + A + B + ... + Z .

Trong SNOBOL, các hằng số số học có thể được biểu diễn như sau :

(+) (digit + (digit * +) +   digit+)

Một số công cụ phát sinh bộ phân tích từ vựng nhận input như một dãy các biểu thức chính quy mô tả các ký hiệu từ vựng và phát sinh một ôtômát hữu hạn đơn giản nhận dạng mọi ký hiệu từ vựng. Thông thường, chúng chuyển đổi biểu thức chính quy thành một NFA với -dịch

chuyển và sau đó xây dựng tập hợp con các trạng thái để có thể phát sinh DFA một cách trực tiếp hơn là tìm cách loại bỏ các phép chuyển nhãn . Mỗi trạng thái kết thúc xác định ký hiệu từ vựng cụ thể đã tìm thấy. Hàm chuyển của FA sẽ được mã hóa bằng một trong vài cách nhằm chiếm ít không gian hơn so với bảng hàm chuyển tổ chức dưới dạng mảng hai chiều. Bộ phân tích từ vựng được thiết lập bằng cách phát sinh một chương trình cố định thông dịch các bảng mã, cùng với các bảng minh họa cụ thể sự nhận dạng của FA trên các ký hiệu từ vựng (viết dưới dạng các biểu thức chính quy). Bộ phân tích từ vựng dạng này có thể được dùng như một chương trình con độc lập (module) trong một trình biên dịch ngôn ngữ.

Trình soạn thảo văn bản

Hiển nhiên, các trình soạn thảo văn bản hoặc các chương trình tương tự cho phép thay thế một chuỗi bởi mọi chuỗi kết hợp với một biểu thức chính quy cho trước.

Chẳng hạn, trình soạn thảo văn bản ed trong UNIX cho phép một câu lệnh như sau :

/aba*c/

để tìm sự xuất hiện đầu tiên của chuỗi có dạng như trên. Hay câu lệnh :

s/bbb*/b/

cho phép thay thế các chuỗi có dạng bbb* thành chuỗi có một ký tự b.

Hay trong các câu lệnh của MS-DOS và NC, ví dụ câu lệnh :

Del tmp*.???

sẽ cho phép xóa đi tất cả các file với tên tập tin bắt đầu bằng tmp, sau đó là một chuỗi bắt kỳ và có phần mở rộng là 3 ký tự tùy ý. Dấu * trong trường hợp này ký hiệu cho một chuỗi bất kỳ, còn dấu ? ký hiệu cho một

ký tự tùy ý. Đây cũng là một dạng ký hiệu của biểu thức chính quy thay thế cho chuỗi.

Hay chẳng hạn, một ví dụ về xử lý chuỗi khác được áp dụng cho việc tìm kiếm theo mẫu trên các trang Web.

Trong tất cả các ví dụ trên, ký hiệu * xác định "mọi" biểu thức a1 + a2 + ... + an trong đó các ai là tất cả các ký tự cho phép trong máy tính trừ ký tự xuống dòng (newline). Ta có thể chuyển một biểu thức chính quy r sang DFA chấp nhận mọi r. Chú ý rằng sự hiện diện của ký hiệu * sẽ cho phép ta nhận dạng một thành phần của L(r) bắt đầu từ bất kỳ vị trí nào trong dòng. Để làm được điều này, các ứng dụng phải thực hiện quá trình chuyển đổi từ một biểu thức chính quy sang NFA. Và vì cơ chế hoạt động của NFA khá phức tạp nên thông thường ngay sau đó, NFA lại phải được biến đổi tiếp thành dạng DFA tương đương. Tuy nhiên, sự chuyển đổi từ một biểu thức chính quy sang DFA tốn nhiều thời gian hơn việc sử dụng DFA để kiểm tra các mẫu bằng cách duyệt qua chúng một lần, tuy DFA có thể có số trạng thái là hàm mũ của độ dài biểu thức chính quy.

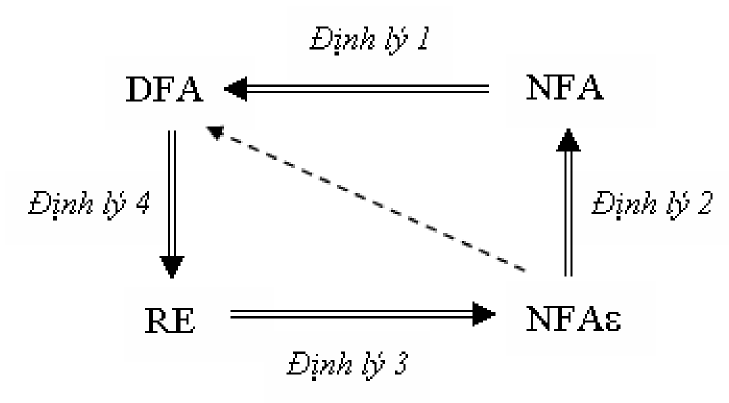
Thực tế, trong trình soạn thảo văn bản UNIX, biểu thức chính quy với mọi r được chuyển thành một NFA có -dịch chuyển và sau đó NFA này được mô phỏng một cách trực tiếp.

Câu hỏi :

?

Hãy tự liên hệ một số các ứng dụng thực tế khác dùng cơ chế ôtômát hữu hạn ?

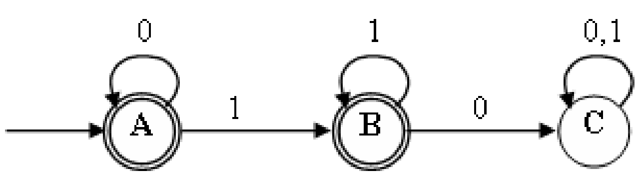
Tổng kết chương III: Phần nội dung chương III tập trung nghiên cứu cơ chế hoạt động của các dạng ôtômát hữu hạn, mối tương quan giữa chúng, cũng như sự tương đương của chúng với biểu thức chính quy. Tùy theo các yêu cầu thực tế của ứng dụng, ta có thể chuyển từ dạng phức tạp nhất sang các dạng đơn giản hơn. Có thể tóm tắt sự tương quan giữa các Định lý trong chương này theo sơ đồ sau :



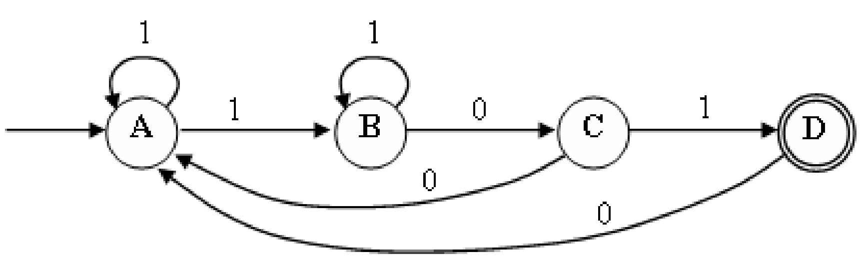
BÀI TẬP CHƯƠNG III

3.1. Mô tả ngôn ngữ được chấp nhận bởi các ôtômát hữu hạn với sơ đồ chuyển được cho như sau :

a)



b)



3.2. Xây dựng các sơ đồ chuyển ôtômát hữu hạn chấp nhận các ngôn ngữ sau trên bộ chữ cái = {0, 1}

1. Tập các chuỗi kết thúc là 00.

2. Tập các chuỗi có 3 ký hiệu 0 liên tiếp.

c) Tập các chuỗi mà ký hiệu thứ 3 kể từ cận phải của chuỗi là 1.

d) Tập mọi chuỗi mà bất cứ chuỗi con nào có độ dài bằng 5 đều có chứa ít nhất 2 con số 0.

3.3. Tìm các sơ đồ chuyển ôtômát hữu hạn đoán nhận các ngôn ngữ sau :

a) Tập các chuỗi trên {0, 1} có chứa một số chẵn các số 0 và một số lẻ các số 1

b) Tập các chuỗi trên {0, 1} có độ dài chia hết cho 3.

c) Tập các chuỗi trên {0, 1} không chứa 101 như một chuỗi con.

3.4. Xây dựng DFA tương đương với mỗi NFA sau :

a) N1({0,1,2,3},{a,b}, 1,0,{3}) với 1 b) N2 ({0,1,2,3}, {a,b}, 2,0, {1,3}) với 2

1 a b 2 a b

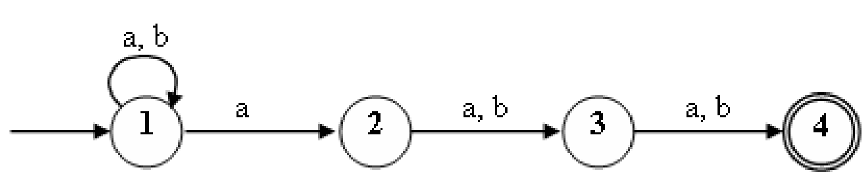
0 {0, 1}{1}0 {1, 3}{1}

1 {2}{2}1 {2}{1, 2}

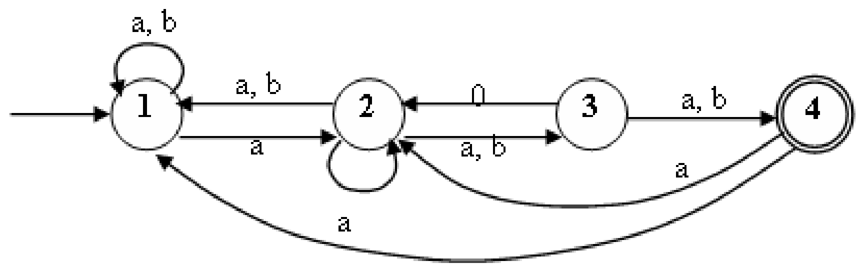
2 {3} 2 {3}{0}

3 {3}{3}3 {0}

c)

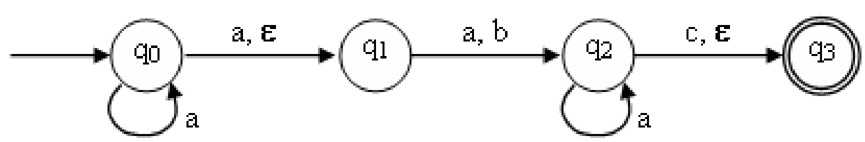


d)

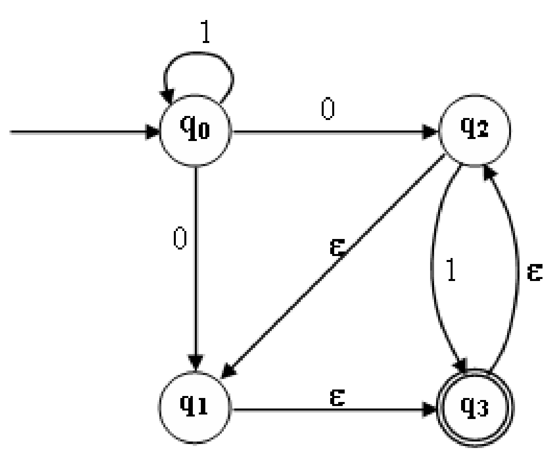


3.5. Tìm NFA không có -dịch chuyển nhận dạng cùng ngôn ngữ với các NFA sau :

a)



b)



3.6. Viết biểu thức chính quy và vẽ NFA đoán nhận các ngôn ngữ sau :

1. Tập hợp các chuỗi trên = {1, 2, 3} sao cho ký hiệu cuối cùng đã có xuất hiện trước đó .
2. Tập hợp các chuỗi trên = {0, 1} trong đó có một cặp ký tự 0 cách nhau bởi một chuỗi con có độ dài 4i, với i 0 nào đó.

3.7. Viết biểu thức chính quy cho mỗi ngôn ngữ sau trên = { 0, 1} :

a) Tập hợp các chuỗi trong đó mọi cặp 0 liên tiếp đều xuất hiện trước mọi cặp 1 liên tiếp.

b) Tập hợp các chuỗi chứa nhiều nhất một cặp 0 liên tiếp và nhiều nhất một cặp 1 liên tiếp.

3.8. Mô tả (bằng lời) ngôn ngữ được các biểu thức chính quy sau đặc tả :

1. 0(0 + 1)* 0
2. (0+ 1)*0(0 + 1) (0 + 1)
3. (11+ 0)*(00+ 1)
4. (1+ 01+ 001)*(+ 0 + 00)
5. [00 + 11 + (01+ 10) (00+ 11)*(01+ 10)]*

3.9. Chứng tỏ các biểu thức chính quy sau ký hiệu cho cùng một ngôn ngữ :

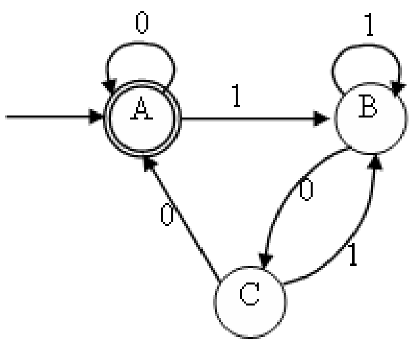
(aa)*,(aa)* + (a),(aa + aaaa)*,(aa)* (aa)*

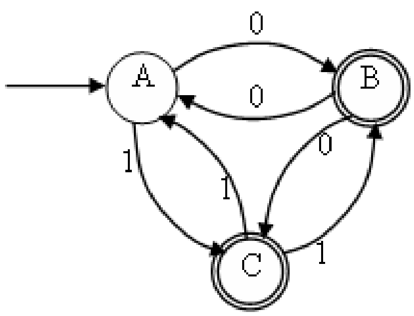
3.10. Vẽ NFA với -dịch chuyển được cho bởi các biểu thức chính qui sau. Sau đó, hãy chuyển sang DFA tương đương :

1. (a* + b*)*
2. ((+ a) b*)*
3. (a + b)* abb (a + b)*
4. ab + (a + bb) a*b
5. (a + ab + aab)*(+ a+ aa)
6. 10 + (0 + 11)0*1
7. 01 [((10)*+ 111)* + 0]*1

3.11. Hãy tìm các biểu thức chính qui tương ứng với các sơ đồ chuyển trạng thái sau:

a)b)





BÀI TẬP LẬP TRÌNH

3.12. Viết chương trình trong Pascal / C mô phỏng một FA chấp nhận ngôn ngữ được biểu diễn bởi biểu thức chính quy sau :

[A ... Z, a ... Z]+ [A ... Z, a ... Z, 0 ... 9, _]* + [0 ... 9]+ + op

3.13. Viết chương trình cho ra một FA tương ứng khi đầu vào là một biểu thức chính quy.

3.14. Viết chương trình cho ra DFA khi đầu vào là một NFA.

Văn phạm phi ngữ cảnh

Nội dung chính : Trong chương này, ta sẽ nghiên cứu một loại văn phạm khá quan trọng, gọi là văn phạm phi ngữ cảnh (CFG) và lớp ngôn ngữ mà chúng mô tả - ngôn ngữ phi ngữ cảnh (CFL). CFL, cũng như tập hợp chính quy, có nhiều ứng dụng thực tế rất quan trọng, đặc biệt trong việc biểu diễn ngôn ngữ lập trình. Chẳng hạn, CFG dùng hữu ích để mô tả các biểu thức số học trong các dấu ngoặc lồng nhau hay những cấu trúc khối trong ngôn ngữ lập trình (cấu trúc khối begin-end). Sau khi định nghĩa văn phạm phi ngữ cảnh, một số cách biến đổi văn phạm phi ngữ cảnh nhằm giản lược nó và đưa nó về một trong những dạng chuẩn sẽ được trình bày. Cuối chương, bổ đề bơm cho ngôn ngữ CFL và một số tính chất nhằm xác định tập ngôn ngữ này cũng sẽ được giới thiệu. Mục tiêu cần đạt: Cuối chương, sinh viên cần phải nắm vững: Khái niệm CFG, xác định các thành phần của một CFG. Nhận dạng được lớp ngôn ngữ mà một văn phạm CFG đặc tả. Xây dựng các luật sinh cho một CFG đặc tả một lớp ngôn ngữ. Các bước giản lược văn phạm CFG không chứa các giá trị vô ích. Chuẩn hóa CFG về các dạng chuẩn Chomsky hoặc Greibach. Ứng dụng bổ đề bơm cho CFL để chứng tỏ một ngôn ngữ không là ngôn ngữ phi ngữ cảnh. Xác định một ngôn ngữ có thuộc lớp ngôn ngữ phi ngữ cảnh hay không theo các tính chất của CFL. Kiểm tra tính rỗng, hữu hạn hoặc vô hạn của một CFL. Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, trước hết sinh viên cần hiểu rõ cấu trúc cú pháp của một số ngôn ngữ lập trình cấp cao như Pascal, C; nắm vững lý thuyết đồ thị và cây; phương pháp chứng minh phản chứng và sự phân cấp các lớp văn phạm theo Noam Chomsky; … Tài liệu tham khảo : [1] John E. Hopcroft, Jeffrey D.Ullman – Introduction to Automata Theory, Languages and Computation – Addison – Wesley Publishing Company, Inc – 1979 (Chapter 4 : Context – Free Grammars). [2] V.J. Rayward-Smith – A First course in Formal Language Theory (Second Editor) – McGraw-Hill Book Company Europe – 1995 (Chapter 5: Context-Free Languages) [3] From Wikipedia, the free encyclopedia – Context-Free Grammar: http://en.wikipedia.org/wiki/Context-free_grammar

VĂN PHẠM PHI NGỮ CẢNH (CFG : Context Free Grammar)

Xuất xứ của văn phạm phi ngữ cảnh là sự mô tả thông qua các ngôn ngữ tự nhiên. Ta có thể viết các quy tắc cú pháp để diễn tả câu “An là sinh viên giỏi“ như sau :

< câu đơn >   < chủ ngữ > < vị ngữ >

< chủ ngữ >   < danh từ >

< vị ngữ >   < động từ > < bổ ngữ >

< bổ ngữ >   < danh từ > < tính từ >

< danh từ >   An

< danh từ >   sinh viên

< động từ >   là

< tính từ >   giỏi

Các từ trong dấu móc nhọn như < câu đơn >, < chủ ngữ >, < vị ngữ >, ... là các phạm trù cú pháp, cho ta vai trò của các bộ phận hợp thành câu. Ta thấy một câu sinh ra qua các bước triển khai dần dần theo các quy tắc cú pháp. Đây cũng chính là dạng của các luật sinh trong văn phạm phi ngữ cảnh. Và như vậy, văn phạm phi ngữ cảnh cũng có thể chọn làm mô hình cho các văn phạm của các ngôn ngữ tự nhiên.

Tuy nhiên, trong khoa học máy tính, với nhu cầu biểu diễn các ngôn ngữ lập trình, văn phạm phi ngữ cảnh CFG còn được thiết kế thành một dạng tương đương gọi là văn phạm BNF (Backus - Naur Form). Đây cũng là văn phạm CFG với những thay đổi nhỏ về dạng thức và một số ký hiệu viết tắt mà các nhà khoa học máy tính thường ứng dụng trong việc diễn tả cú pháp của các ngôn ngữ lập trình cấp cao (như ALGOL, PASCAL, ...). Trong dạng thức của văn phạm BNF, ký hiệu ::= được dùng thay cho

ký hiệu . Chẳng hạn, để định nghĩa một biểu thức số học (expression) bao gồm các danh biểu (identifier) tham gia vào các phép toán +, * hoặc biểu thức con lồng trong dấu ngoặc đơn , ta viết :

<expression> ::= <expression> + <expression>

<expression> ::= <expression> * <expression>

<expression> ::= (<expression>)

<expression> ::= <identifier>

Việc nghiên cứu các văn phạm phi ngữ cảnh đã tạo nên một cơ sở lý luận vững chắc cho việc biểu diễn ngôn ngữ lập trình, việc tìm kiếm các giải thuật phân tích cú pháp vận dụng trong chương trình dịch và cho nhiều ứng dụng khác về xử lý chuỗi. Chẳng hạn, nó rất hữu ích trong việc mô tả các biểu thức số học với nhiều dấu ngoặc lồng nhau hoặc cấu trúc khối trong ngôn ngữ lập trình mà biểu thức chính quy không thể đặc tả.

Định nghĩa

Văn phạm phi ngữ cảnh là một tập hợp hữu hạn các biến (còn gọi là các ký hiệu chưa kết thúc), mỗi biến biểu diễn một ngôn ngữ. Ngôn ngữ được biểu diễn bởi các biến được mô tả một cách đệ quy theo thuật ngữ của một khái niệm khác gọi là ký hiệu kết thúc. Quy tắc quan hệ giữa các biến gọi là luật sinh. Mỗi luật sinh có dạng một biến ở vế trái sinh ra một chuỗi có thể gồm biến lẫn các ký hiệu kết thúc trong văn phạm.

Văn phạm phi ngữ cảnh (CFG) là một hệ thống gồm bốn thành phần, ký hiệu là văn phạm G (V, T, P, S), trong đó :

. V là tập hữu hạn các biến (hay ký tự chưa kết thúc)

. T là tập hữu hạn các ký tự kết thúc, V   T =

. P là tập hữu hạn các luật sinh mà mỗi luật sinh có dạng A   với A là biến và   là chuỗi các ký hiệu   (V   T)*

. S là một biến đặc biệt gọi là ký hiệu bắt đầu văn phạm.

Thí dụ 5.1 : Văn phạm G ({S, A, B}, {a, b}, P, S), trong đó P gồm các luật sinh sau:

S ® AB

A ® aA

A ® a

B ® bB

B ® b

Quy ước ký hiệu:

- Các chữ in hoa A, B, C, D, E, ... và S ký hiệu các biến (S thường được dùng làm ký hiệu bắt đầu).

- Các chữ nhỏ a, b, c, d, e, ...; các chữ số và một số ký hiệu khác ký hiệu cho các ký hiệu kết thúc.

- Các chữ in hoa X, Y, Z là các ký hiệu có thể là ký hiệu kết thúc hoặc biến.

- Các chữ Hi-lạp , , , ... biểu diễn cho chuỗi các ký hiệu kết thúc và biến.

Ta sẽ biểu diễn văn phạm một cách tóm tắt bằng cách chỉ liệt kê các luật sinh của nó. Nếu A ® 1, A ® 2 , ... , A ® k là các luật sinh của biến A trong văn phạm nào đó, ta sẽ ghi ngắn gọn là A ® 1 | 2 | ... | k

Thí dụ 5.2 : Văn phạm trong Thí dụ 5.1 trên có thể viết gọn là :

S ® AB

A ® aA a

B ® bB b

Câu hỏi :

?

Bạn nghĩ gì về lớp ngôn ngữ có thể được sinh bởi văn phạm trong ví dụ trên ? Cơ chế nào có thể được sử dụng cho văn phạm để phát sinh ngôn ngữ ?

Dẫn xuất và ngôn ngữ

Dẫn xuất: Để định nghĩa ngôn ngữ sinh bởi văn phạm CFG G (V, T, P, S), ta dẫn nhập khái niệm dẫn xuất. Trước hết ta giới thiệu hai quan hệ G và *G giữa hai chuỗi trong tập (V T)*. Nếu A là một luật sinh trong văn phạm và , là hai chuỗi bất kỳ trong tập (V T)* thì A G , hay ta còn nói luật sinh A áp dụng vào chuỗi A để thu được chuỗi , nghĩa là A sinh trực tiếp trong văn phạm G. Hai chuỗi gọi là quan hệ nhau bởi G nếu chuỗi thứ hai thu được từ chuỗi thứ nhất bằng cách áp dụng một luật sinh nào đó.

Giả sử 1, 2, ..., m là các chuỗi thuộc (V T)* với m 1 và :

1 G 2, 2 G 3, …, m -1 G m

thì ta nói 1 *G m hay 1 dẫn xuất ra m trong văn phạm G.

Như vậy, *G là bao đóng phản xạ và bắc cầu của G. Nói cách khác, *G nếu được dẫn ra từ bằng không hoặc nhiều hơn các luật sinh của P. Chú ý rằng *G với mọi chuỗi .

Thông thường nếu không có nhầm lẫn ta sẽ dùng các ký hiệu   và   * thay cho ký hiệu   G và   *G. Nếu   dẫn ra   bằng i bước dẫn xuất thì ta ký hiệu   i   .

Ngôn ngữ sinh bởi văn phạm phi ngữ cảnh

Cho văn phạm CFG G(V, T, P, S), ta định nghĩa :

L(G) = {w w   T * và S   *G w}

Nghĩa là, một chuỗi thuộc L(G) nếu:

1) Chuỗi gồm toàn ký hiệu kết thúc.

2) Chuỗi được dẫn ra từ ký hiệu bắt đầu S.

Ta gọi L là ngôn ngữ phi ngữ cảnh (CFL) nếu nó là L(G) với một CFG G nào đó. Chuỗi   gồm các ký hiệu kết thúc và các biến, được gọi là một dạng câu sinh từ G nếu S   * . Hai văn phạm G1, G2 được gọi là tương đương nếu L(G1) = L(G2)

Thí dụ 5.3 : Xét văn phạm G (V, T, P, S), trong đó :

V = {S}, T = {a, b}, P = {S ® aSb, S ® ab}.

Bằng cách áp dụng luật sinh thứ nhất n -1 lần và luật sinh thứ hai 1 lần, ta có:

S Þ aSb Þ aaSbb Þ a3Sb3 Þ ... Þ an-1bn-1 Þ anbn

Vậy, L(G) chứa các chuỗi có dạng anbn, hay L(G) = {anbn   n   1}.

Cây dẫn xuất

Để dễ hình dung sự phát sinh ra các chuỗi trong văn phạm phi ngữ cảnh, ta thường diễn tả một chuỗi dẫn xuất qua hình ảnh một cây. Một cách

hình thức, ta định nghĩa như sau:

Định nghĩa : Cho văn phạm G (V, T, P, S). Cây dẫn xuất (hay cây phân tích cú pháp) của G được định nghĩa như sau :

i) Mỗi nút (đỉnh) có một nhãn, là một ký hiệu (V T { })

ii) Nút gốc có nhãn là ký hiệu bắt đầu S.

iii) Nếu nút trung gian có nhãn A thì A V

iv) Nếu nút n có nhãn A và các đỉnh n1, n2, ..., nk là con của n theo thứ tự từ trái sang phải có nhãn lần lượt là X1, X2, ..., Xk thì A X1X2 ... Xk là một luật sinh trong tập luật sinh P.

v) Nếu nút n có nhãn là từ rỗng thì n phải là nút lá và là nút con duy nhất của nút cha của nó.

Thí dụ 5.4 : Xét văn phạm G ({S, A}, {a, b}, P, S), trong đó P gồm:

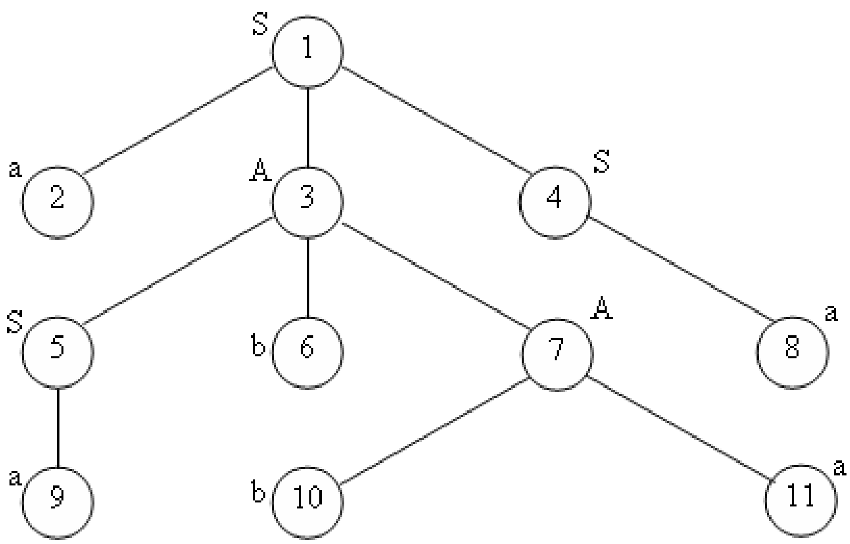
S ® aAS | a

A ® SbA | SS | ba

Một cây dẫn xuất từ văn phạm có dạng như hình 5.1 sau :

Ta thấy, nút 1 có nhãn S và các con của nó lần lượt là a, A, S (chú ý S aAS là một luật sinh). Tương tự, nút 3 có nhãn A và các con của nó là S, b, A (từ luật sinh A SbA). Nút 4, 5 có cùng nhãn S và có nút con nhãn a (luật sinh S a). Cuối cùng nút 7 có nhãn A và có các nút con b, a (luật sinh A ba).

Trên cây dẫn xuất, nếu ta đọc các lá theo thứ tự từ “trái sang phải“ thì ta có một dạng câu trong G. Ta gọi chuỗi này là chuỗi sinh bởi cây dẫn xuất.



Hình 5.1 - Cây dẫn xuất từ văn phạm

Một cây con (subtree) của cây dẫn xuất có nút gốc nhãn là A còn được gọi là A-cây con (hoặc A-cây). Cây con cũng giống như cây dẫn xuất, chỉ khác là nhãn của nút gốc không nhất thiết phải là ký hiệu bắt đầu S.

Thí dụ 5.5 :Xét văn phạm và cây dẫn xuất trong Hình 5.1. Đọc các lá theo thứ tự từ trái sang phải ta có chuỗi aabbaa, trong trường hợp này tất cả các lá đều là ký hiệu kết thúc, nhưng nói chung cũng không bắt buộc như thế, lá có thể có nhãn là   hoặc biến. Ta thấy dẫn xuất S   * aabbaa bằng chuỗi dẫn xuất :

S Þ aAS Þ aSbAS Þ aabAS Þ aabbaS Þ aabbaa

A-cây có nút đỉnh là 3 tạo ra chuỗi con abba theo chuỗi dẫn xuất :

S Þ SbAÞ abA Þ abba

Câu hỏi :

?

Các cây dẫn xuất được sinh từ những chuỗi dẫn xuất khác nhau cho cùng một chuỗi nhập có là những cây dẫn xuất khác nhau không ?

Quan hệ giữa dẫn xuất và cây dẫn xuất

ĐỊNH LÝ 5.1 : Nếu G (V, T, P, S) là một văn phạm phi ngữ cảnh thì S * nếu và chỉ nếu có cây dẫn xuất trong văn phạm sinh ra .

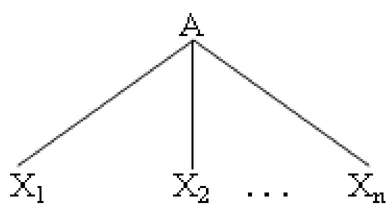
Chứng minh

Ta chứng minh rằng với biến A bất kỳ, A * nếu và chỉ nếu có một A-cây sinh ra .

Nếu: Giả sử được sinh bởi A-cây, ta chứng minh quy nạp theo số nút trung gian của cây dẫn xuất rằng A * .

Nếu có 1 nút trung gian thì cây phải có dạng như hình sau :

Khi đó X1X2 ... Xn là chuỗi và A là một luật sinh trong P theo định nghĩa cây dẫn xuất.



Hình 5.2(a) - A-cây với một nút trong

Giả sử kết quả đúng tới k -1 nút trung gian (k > 1)

Ta chứng minh kết quả cũng đúng với k nút.

Xét được sinh ra bởi A-cây có k nút trung gian. Rõ ràng các nút con của nút gốc không phải tất cả đều là lá, ta gọi chúng từ trái sang phải là X1, X2, ..., Xn thì chắc chắn rằng A X1X2 ... Xn là một luật sinh. Xét nút Xi bất kỳ :

- Nếu Xi không là nút lá thì Xi phải là một biến và Xi - cây con sẽ sinh ra một chuỗi i nào đó.

- Nếu Xi là nút lá, ta đặt i = Xi. Dễ thấy rằng nếu j < i thì các j ở bên trái j, do vậy chuỗi đọc từ lá vẫn có dạng = 1 2 ... n. Mỗi Xi - cây con phải có ít nút trung gian hơn cây ban đầu, vì thế theo giả thiết quy nạp, với mỗi đỉnh i không phải là lá thì Xi * i.

Vậy A X1X2 ... Xn * 1X2 ... Xn * 1 2X3 ... Xn * ... * 1 2 ... n =

Hay ta có A * . Chú ý rằng đây chỉ là một trong nhiều cách dẫn xuất ra .

Chỉ nếu : Ngược lại, giả sử A * ta cần chỉ ra một A - cây sinh ra .

Nếu A * bằng một bước dẫn xuất thì A là một luật sinh trong P và có cây dẫn xuất sinh ra như trong hình trên.

Giả sử kết quả đúng tới k-1 bước dẫn xuất

Xét A * bằng k bước dẫn xuất, gọi bước đầu tiên là A X1X2 ... Xn.

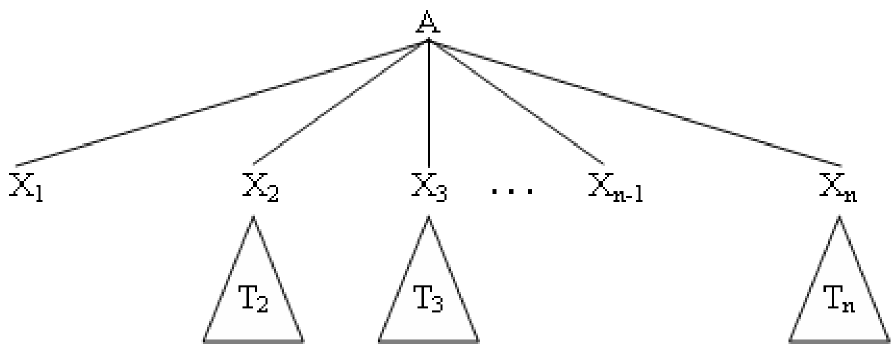
Rõ ràng, một ký hiệu trong phải được dẫn ra từ một biến Xi nào đó. Vì vậy, ta có thể viết = 1 2 ... n, trong đó mỗi 1 i n thoả mãn :

- i = Xi nếu Xi là ký hiệu kết thúc.

- Xi * i nếu Xi là một biến.

Nếu Xi là biến thì dẫn xuất của i từ Xi phải có ít hơn k bước. Vì vậy, theo giả thiết quy nạp ta có Xi - cây sinh ra i, đặt cây này là Ti

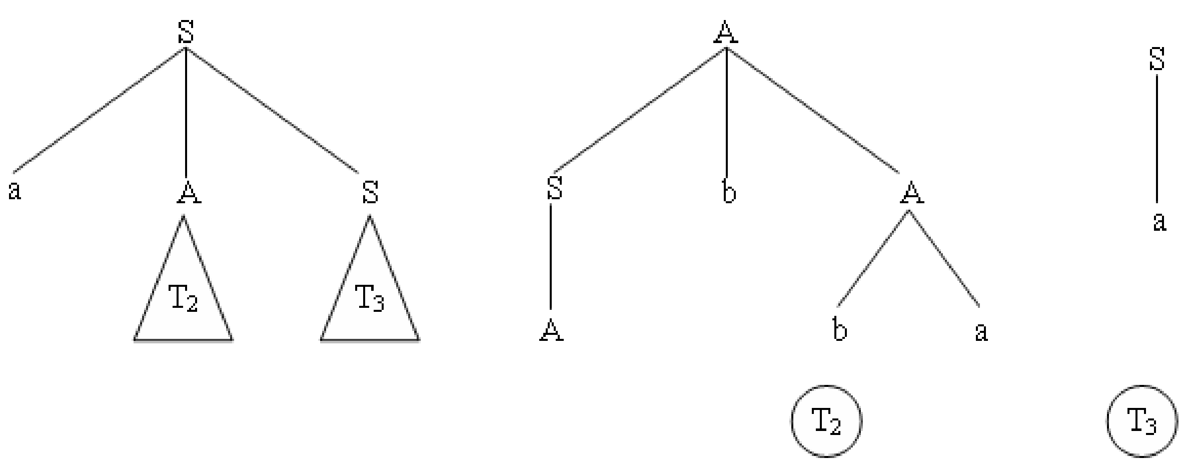
Bây giờ ta dựng A - cây có n lá X1X2 ... Xn. Mỗi Xi không là ký hiệu kết thúc ta thay bằng cây Ti tương ứng. Cuối cùng, ta có cây dẫn xuất sinh ra có dạng như sau :



Hình 5.2(b) - A-cây

Thí dụ 5.6 :Xét chuỗi dẫn xuất S   * aabbaa cho văn phạm ở Thí dụ 5.4.

Bước đầu tiên trong dẫn xuất đó là S   aAS. Theo dõi các bước suy dẫn sau đó, ta thấy biến A được thay bởi SbA, rồi trở thành abA và cuối cùng thành abba, đó chính là kết quả của cây T2 (A - cây). Còn biến S thì được thay bởi a và đó là kết quả của cây T3 (S -cây). Ghép nối lại, ta được cây dẫn xuất mà kết quả là chuỗi aabbaa như dưới đây.



[

Hình 5.3 - Ghép nối các cây dẫn xuất

Dẫn xuất trái nhất, dẫn xuất phải nhất

Nếu tại mỗi bước dẫn xuất, luật sinh được áp dụng vào biến bên trái nhất thì ta gọi đó là dẫn xuất trái nhất (leftmost) hay dẫn xuất trái. Tương tự, nếu biến bên phải nhất được thay thế ở mỗi bước dẫn xuất, đó là dẫn xuất phải nhất (rightmost) hay dẫn xuất phải. Nếu chuỗi w L(G) với CFG G thì w sẽ có ít nhất một cây dẫn xuất ra nó và tương ứng với các cây này, w chỉ có duy nhất một dẫn xuất trái nhất và duy nhất một dẫn xuất phải nhất. Dĩ nhiên, w có thể có nhiều dẫn xuất trái (phải) nhất vì nó có thể có nhiều cây dẫn xuất.

Thí dụ 5.7 : Xét cây dẫn xuất ở Hình 5.1

. Dẫn xuất trái nhất của cây :

S Þ aAS Þ aSbAS Þ aabAS Þ aabbaS Þ aabbaa.

. Dẫn xuất phải nhất tương ứng là :

S Þ aAS Þ aAa Þ aSbAa Þ aSbbaa Þ aabbaa.

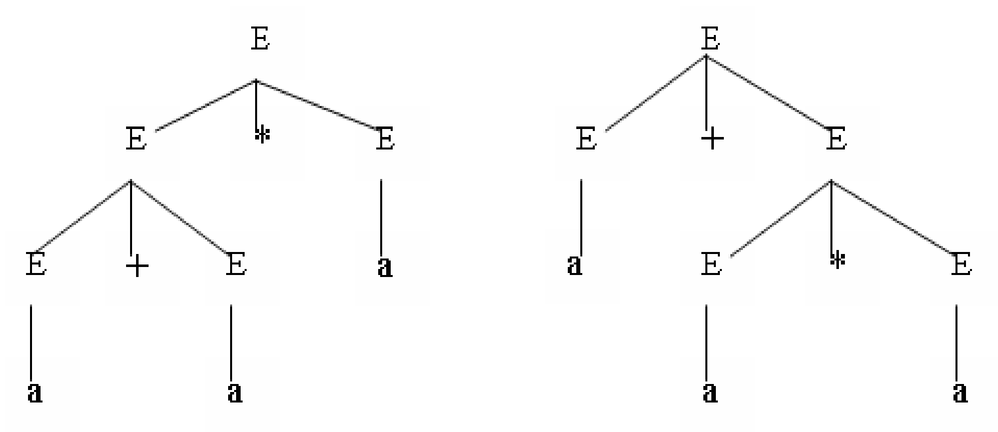
Văn phạm mơ hồ

Một văn phạm phi ngữ cảnh G có nhiều hơn một cây dẫn xuất cho cùng một chuỗi w, thì G được gọi là văn phạm mơ hồ (ambiguity). Dĩ nhiên, cũng có thể nói rằng văn phạm G là mơ hồ nếu có một chuỗi w được dẫn ra từ ký hiệu bắt đầu S với hai dẫn xuất trái hoặc hai dẫn xuất phải.

Thí dụ 5.8 : Xét văn phạm G với các luật sinh như sau :

E   E + E   E * E   (E)   a

Văn phạm này sinh ra các chuỗi biểu thức số học với 2 phép toán + và * . Với chuỗi a + a * a, ta có thể vẽ đến hai cây dẫn xuất khác nhau như sau :



(a) (b)

Hình 5.4 - Các cây dẫn xuất khác nhau cho cùng chuỗi nhập

Điều này có nghĩa là biểu thức a + a * a có thể hiểu theo hai cách khác nhau: thực hiện phép cộng trước hay phép nhân trước ? Để khắc phục sự mơ hồ này, ta có thể :

- Hoặc quy định rằng các phép cộng và nhân luôn luôn được thực hiện theo thứ tự từ trái sang phải (trừ khi gặp ngoặc đơn). Ta viết văn phạm G1 không mơ hồ tương đương như sau :

E E + T E * T T

T (E) a

- Hoặc quy định rằng khi không có dấu ngoặc đơn ngăn cách thì phép nhân luôn luôn được ưu tiên hơn phép cộng. Ta viết văn phạm G2 không mơ hồ tương đương như sau :

E E + T T

T T * F F

F (E) a

GIẢN LƯỢC CÁC VĂN PHẠM PHI NGỮ CẢNH

Thường thì một văn phạm phi ngữ cảnh có thể còn chứa đựng một vài yếu tố thừa, vô ích. Chẳng hạn như theo các đặc tính trên, có những ký hiệu không thực sự tham gia vào quá trình dẫn xuất ra câu, hoặc sẽ có những luật sinh dạng A B làm kéo dài chuỗi dẫn xuất một cách không cần thiết. Vì vậy, việc giản lược văn phạm phi ngữ cảnh là nhằm loại bỏ những yếu tố vô ích đó mà không làm giảm bớt khả năng sản sinh ngôn ngữ của văn phạm.

Nếu L là một CFL, nó có thể tạo ra văn phạm CFG với các đặc tính sau :

1. Mỗi biến và mỗi ký hiệu kết thúc của G đều xuất hiện trong dẫn xuất của một số chuỗi trong L.
2. Không có luật sinh nào dạng A B, mà trong đó A, B đều là biến.

Hơn nữa, nếu L thì không cần luật sinh A . Thực tế, nếu L, ta có mọi luật sinh trong G đều có một trong hai dạng :

A BC hoặc A a (là chuỗi các biến hoặc)

A a

Hai dạng đặc biệt này gọi là dạng chuẩn Chomsky và dạng chuẩn Greibach.

Các ký hiệu vô ích

Một ký hiệu X gọi là có ích nếu có một dẫn xuất S * X * w với các chuỗi , bất kỳ và w T *. Ngược lại X gọi là vô ích.

Vậy, có 2 đặc điểm cho ký hiệu có ích:

- X phải dẫn ra một chuỗi ký hiệu kết thúc.

- X phải nằm trong dẫn xuất từ S.

Tuy nhiên 2 dấu hiệu trên không đủ để đảm bảo X có ích vì X có thể nằm trong dạng câu chứa một biến nhưng từ đó không có ký hiệu kết thúc được sinh ra.

BỔ ĐỀ 1: (Dùng loại bỏ các biến không dẫn ra chuỗi ký hiệu kết thúc)

Cho CFG G (V, T, P, S) với L(G) , có một CFG G’ (V’, T’, P’, S) tương đương sao cho mỗi A V’ tồn tại w T* để A * w.

Chứng minh

Mỗi biến A với luật sinh A w trong P thì rõ ràng A V’. Nếu A X1X2 ... Xn là một luật sinh, trong đó mỗi Xi hoặc là ký hiệu kết thúc hoặc là một biến đã có sẵn trong V’ thì một chuỗi các ký hiệu kết thúc có thể được dẫn ra từ A bằng dẫn xuất bắt đầu A X1X2 ... Xn, vì vậy A V’. Tập V’ có thể tính được bằng cách lặp lại giải thuật trên. P’ là tập tất cả các luật sinh mà các ký hiệu của nó thuộc V’ T.

Giải thuật tìm V' như sau:

Begin

(1) OLDV:= Æ;

(2) NEWV:= {A A w với w T *};

(3) While OLDV ¹ NEWV do

begin

(4) OLDV := NEWV;

(5) NEWV := OLDV {A A   với   (T OLDV)*}

end;

(6) V' := NEWV;

end;

Rõ ràng rằng nếu biến A được thêm vào V’ tại bước (2) hoặc (5) thì A sẽ dẫn ra được chuỗi ký hiệu kết thúc. Ta chứng minh rằng nếu A dẫn ra được một chuỗi ký hiệu kết thúc thì A được thêm vào tập NEWV.

Dùng chứng minh quy nạp theo độ dài của dẫn xuất A * w.

Nếu độ dài bằng 1 thì A là một luật sinh trong P. Vậy A được đưa vào V’ tại bước (2).

Giả sử kết quả đúng tới k -1 bước dẫn xuất (k >1)

Nếu A X1X2 ... Xn * w bằng k bước thì ta có thể viết w = w1w2 ... wn, trong đó Xi * wi, với 1 i n bằng ít hơn k bước dẫn xuất. Theo giả thiết quy nạp thì các biến Xi này được thêm vào V’. Khi Xi cuối cùng được thêm vào V’ thì vòng lặp (3) vẫn tiếp tục lặp một lần nữa và A sẽ được thêm vào V’ tại (5).

Ta chứng minh L(G’) = L(G) :

Chọn V’ là tập hợp tại (6) và P' là tập tất cả các luật sinh mà các ký hiệu của nó thuộc (V’ T) thì chắc chắn rằng có tồn tại văn phạm G’ (V’, T, P’, S) thoả mãn tính chất: nếu A V’ thì A * w với w nào đó thuộc T *. Hơn nữa, mỗi luật sinh của P’ đều là luật sinh của P nên ta có L(G’) L(G).

Ngược lại giả sử một từ w L(G) - L(G’) thì một dẫn xuất bất kỳ của w phải liên quan đến các biến thuộc V – V’ hoặc luật sinh thuộc P – P’ (các dẫn xuất này đưa ra các biến thuộc V – V’), nhưng do không có biến nào trong V – V’ dẫn đến chuỗi kết thúc, điều này dẫn đến mâu thuẫn.

Vậy L(G’) = L(G).

Hay có thể nói 2 ngôn ngữ được cho từ 2 văn phạm G và G’ là tương đương nhau, hay nói cách khác: nếu có một văn phạm G thì luôn luôn có

một văn phạm G’ tương ứng mà trong đó mỗi biến của G’ đều cho ra ký hiệu kết thúc.

BỔ ĐỀ 2: (Dùng loại bỏ các biến không được dẫn ra từ ký hiệu bắt đầu văn phạm)

Nếu G (V, T, P, S) là CFG thì ta có thể tìm được CFG G’ (V’, T’, P’, S) tương đương sao cho mỗi X V’ T’ tồn tại , (V’ T’)* để S * X .

Chứng minh

Tập V’ T’ gồm các ký hiệu xuất hiện trong dạng câu của G được xây dựng bởi giải thuật lặp như sau :

. Đặt V’ = {S}; T’ = ;

. Nếu A V’ và A 1| 2 | ... | n là các luật sinh trong P thì thêm tất cả các biến của 1, 2, ... , n vào V’ và các ký hiệu kết thúc của 1, 2 ,..., n vào T’.

. Lặp lại giải thuật cho đến khi không còn biến hoặc ký hiệu kết thúc nào được thêm vào nữa.

Dễ thấy, X V’ T’ thì tồn tại , (V’ T’)* để S * X , trong đó P’ là tập hợp tất cả các luật sinh của P chỉ chứa các ký hiệu thuộc (V’ T’).

Ta dễ dàng chứng minh L(G’) = L(G) .

ĐỊNH LÝ 5.2: Mỗi ngôn ngữ phi ngữ cảnh (CFL) không rỗng được sinh ra từ một văn phạm phi ngữ cảnh (CFG) không có ký hiệu vô ích.

Chứng minh

Đặt L = L(G) là CFL không rỗng.

Đặt G1 là kết quả của việc áp dụng bổ đề 1 vào G và G2 là kết quả của việc áp dụng bổ đề 2 vào G1.

Giả sử G2 có ký hiệu vô ích là X. Theo bổ đề 2 ta có S *G2 X . Vì tất cả các ký hiệu trong G2 đều có trong G1 nên theo bổ đề 1: S *G1 X *G1 w với w là chuỗi ký hiệu kết thúc. Vì vậy không có ký hiệu nào trong dẫn xuất X *G1 w bị loại bỏ bởi bổ đề 2, vậy X dẫn ra ký hiệu kết thúc trong G2 . Suy ra X là ký hiệu có ích (mâu thuẫn).

Vậy văn phạm G2 không có ký hiệu vô ích nào.

Thí dụ 5.9 : Xét văn phạm có các luật sinh sau :

S ® AB | a

A ® a

Áp dụng bổ đề 1, ta thấy không có ký hiệu kết thúc được nào dẫn ra từ B nên ta loại bỏ B và luật sinh S AB. Tiếp tục, áp dụng bổ đề 2 cho văn phạm :

S ® a

A ® a

Ta thấy chỉ có S xuất hiện trong dạng câu. Vậy ({S}, {a}, {S ® a}, S) là văn phạm tương đương với văn phạm đã cho và không có ký hiệu vô ích.

Câu hỏi :

?

Bạn hãy cho nhận xét về thứ tự áp dụng Bổ đề 1 và Bổ đề 2 trong quá trình loại bỏ các ký hiệu vô ích trong văn phạm ?

Luật sinh

Một luật sinh có dạng A gọi là luật sinh .

Ta xét đến việc loại bỏ các luật sinh này. Nếu L(G) thì không thể loại được tất cả các luật sinh , nhưng nếu L(G) thì có thể. Phương pháp loại bỏ dựa trên việc xác định liệu một biến A có dẫn xuất A * hay không ? Nếu có, ta gọi A là biến rỗng (nullable). Ta có thể thay thế mỗi luật sinh B X1X2 ... Xn bằng tất cả các luật sinh được định dạng bởi việc xóa bỏ tập hợp con các biến Xi rỗng, nhưng không bao gồm luật sinh B , ngay cả khi tất cả các Xi đều là biến rỗng.

ĐỊNH LÝ 5.3 : Nếu L = L(G) với CFG G (V, T, P, S) thì L - { } là L(G') với CFG G' không có ký hiệu vô ích và không có luật sinh .

Chứng minh

Ta có thể xác định tập hợp các biến rỗng (nullable) của G bằng giải thuật lặp như sau : Bắt đầu, nếu A là một luật sinh thì A là biến rỗng. Kế tiếp, nếu B , trong đó gồm toàn các ký hiệu là các biến rỗng đã được tìm thấy trước đó thì B cũng là biến rỗng. Lặp lại cho đến khi không còn biến rỗng nào được tìm thấy nữa.

Tập luật sinh P' được xây dựng như sau : Nếu A X1X2 ... Xn là một luật sinh trong P thì ta thêm tất cả các luật sinh A 1 2 ... n vào P' với điều kiện :

1) Nếu Xi không là biến rỗng thì i = Xi;

2) Nếu Xi là biến rỗng thì i là Xi hoặc ;

3) Không phải tất cả i đều bằng .

Đặt G'' = (V, T, P', S). Ta sẽ chứng minh rằng với mọi A V và w T *, A *G'' w nếu và chỉ nếu w và A *G w.

Nếu: Đặt A iG w và w , ta chứng minh quy nạp rằng A *G'' w.

Nếu i = 1 ta có A w là một luật sinh trong P, và vì w nên luật sinh này cũng thuộc P'.

Giả sử kết quả đúng tới i - 1 (i >1)

Xét A G X1X2 ...Xn i -1G w. Ta viết w = w1w2 ...wn sao cho j, Xj *wj.

Nếu wj và Xj là biến thì theo giả thiết quy nạp, ta có Xj *G'' wj (vì dẫn xuất Xj * wj có ít hơn i bước). Nếu wj = thì Xj là biến rỗng, vậy A 1 2 ... n là một luật sinh trong P', trong đó j = Xj nếu wj và j = nếu wj = .

Vì w nên không phải tất cả j là . Vậy, ta có dẫn xuất :

A Þ 1 2 ... n Þ* w1 2 ... n Þ* w1w2 3 ... n Þ* ... Þ* w1w2 ...wn = w trong G''.

Chỉ nếu: Giả sử A iG'' w. Chắc chắn rằng w vì G'' không có luật sinh . Ta quy nạp theo i rằng A G w.

Nếu i = 1: Ta thấy A w là một luật sinh trong P', do đó cũng phải có luật sinh A w trong P sao cho bằng việc loại bỏ các ký hiệu rỗng trong , ta có w. Vậy, có tồn tại dẫn xuất A G Þ*G w, trong đó Þ* w liên quan đến các dẫn xuất từ các biến rỗng của mà chúng ta đã loại bỏ khỏi w.

Giả sử kết quả đúng tới i - 1 (i >1)

Xét A G'' X1X2 ... Xn i - 1G'' w. Phải có luật sinh A trong P sao cho X1X2 ... Xn tìm được khi loại bỏ các biến rỗng của . Vậy A Þ*G X1X2 ...Xn (chứng minh tương tự như ở trên). Ta viết w = w1w2 ...wn sao cho j ta có Xj *G'' wj (bằng ít hơn i bước). Theo giả thiết quy nạp Xj *G'' wj nếu Xj là biến. Nếu Xj là ký hiệu kết thúc thì wj = Xj và Xj *G wj là hiển nhiên. Vậy A *G w.

Cuối cùng ta áp dụng bổ đề 2 vào G'' ta thu được G' không có ký hiệu vô ích. Vì bổ đề 1 và bổ đề 2 không đưa ra thêm luật sinh mới nào nên G' không có chứa ký hiệu là biến rỗng hay ký hiệu vô ích.

Hơn nữa S   *G’ w nếu và chỉ nếu S   *G w. Vậy L(G’) = L(G) - { }.

Thí dụ 5.10 : Loại bỏ luật sinh e trong văn phạm sau :

S ® AB

A ® aA | e

B ® bB | e

Trước hết, ta xác định tập các biến rỗng trong văn phạm: A, B là các biến rỗng vì có các luật sinh A ® e và B ® e. S cũng là biến rỗng vì có luật sinh S ® AB với A, B đều là các biến rỗng.

Tập biến rỗng Nullable = {A, B, S}

Theo quy tắc xây dựng tập luật sinh P’ trong định lý , ta có tập luật sinh mới như sau :

S ® AB | A | B

A ® aA | a

B ® bB | b

Lưu ý rằng văn phạm mới G’ không sản sinh ra , trong khi G lại có sinh ra từ rỗng e. Vậy muốn có một văn phạm thực sự tương đương với văn phạm G thì ta phải bổ sung thêm luật sinh S ® e vào tập luật sinh của G’. Ta có, văn phạm G’ tương đương G.

Luật sinh đơn vị

Một luật sinh có dạng A ® B với A, B đều là biến gọi là luật sinh đơn vị.

ĐỊNH LÝ 5.4 : Mỗi CFL không chứa e được sinh ra bởi CFG không có ký hiệu vô ích, luật sinh e hoặc luật sinh đơn vị .

Chứng minh

Đặt L là CFL không chứa e và L = L(G) với G (V, T, P, S) là một CFG nào đó.

Theo định lý 3 ta có thể giả sử G không có luật sinh e. Xây dựng tập hợp mới P’ gồm các luật sinh từ P như sau:

Đầu tiên đưa các luật sinh không là luật sinh đơn vị vào P’.

Sau đó, nếu có luật sinh đơn vị dạng A * B với A, B V thì thêm vào P’ tất cả các luật sinh dạng A , với B không phải là luật sinh đơn vị của P.

Chú ý rằng ta có thể dễ dàng kiểm tra có hay không A *G B vì G không có luật sinh e và nếu A G B1 G B2 ... G Bm G B (trong đó một vài biến nào đó có thể xuất hiện 2 lần) thì ta có thể tìm một chuỗi rút ngắn hơn A *G B, vì vậy ta chỉ xét các luật sinh đơn vị không có biến lặp lại.

Bây giờ ta sửa lại văn phạm G’ (V, T, P’, S). Chắc chắn rằng nếu A là một luật sinh trong P’ thì A *G . Vậy nếu có dẫn xuất trong G’ thì có dẫn xuất trong G. Giả sử rằng w L(G). Xét dẫn xuất trái của w trong G:

S G 0 G 1 G ... G n = w.

Nếu 0 i < n thì nếu trong G có i G i+1 bằng luật sinh không là luật sinh đơn vị thì trong G’ cũng có i G’ i+1 không là luật sinh đơn vị. Giả sử i G i+1 bằng luật sinh đơn vị, nhưng bước dẫn xuất trước đó i - 1 i không phải bằng luật sinh đơn vị hoặc i = 0. Và chuỗi dẫn xuất trong G từ i + 1 G i + 2 G ... G j tất cả đều bằng luật sinh đơn vị, còn từ j G j+1 không là luật sinh đơn vị thì ta thấy tất cả các i, i+1, …, j sẽ có cùng độ dài và vì chuỗi dẫn xuất là dẫn xuất trái nên các ký hiệu thay thế phải ở cùng một vị trí. Do vậy, tại vị trí này j G j+1 bằng một luật sinh nào đó thuộc P’- P hay có nghĩa là một luật sinh không thuộc văn phạm G. Điều này sinh ra mâu thuẫn. Vậy L(G) = L(G’).

Ta còn có G’ không có chứa luật sinh đơn vị (theo chứng minh trên) nên G cũng sẽ không chứa luật sinh đơn vị (do G   G’).

Việc áp dụng bổ đề 1, bổ đề 2 để loại các ký hiệu vô ích không đưa ra thêm luật sinh nào chứng tỏ G không chứa ký hiệu vô ích.

Vậy, kết quả ta được một văn phạm thỏa điều kiện định lý.

Thí dụ 5.11 : Loại bỏ các luật sinh đơn vị trong văn phạm sau :

E ® E + T | T

T ® T * F | F

F ® (E) | a

Gọi tập   A = {B   A   * B}, xét các biến trong văn phạm, ta có :

  E = { E, T, F }   T = { T, F }   F = { F }

Vậy tập luật sinh mới, theo định lý sẽ chứa các luật sinh không là luật sinh đơn vị trong P, bổ sung thêm các luật sinh mới thay cho luật sinh đơn vị như sau :

E ® E + T | T * F | (E) | a

T ® T * F | (E) | a

F ® (E) | a

CHUẨN HÓA VĂN PHẠM PHI NGỮ CẢNH

Phần này sẽ giới thiệu hai định lý dùng chuẩn hóa CFG về một trong hai dạng chuẩn Chomsky và Greibach.

Dạng chuẩn Chomsky - CNF (Chomsky Normal Form)

ĐỊNH LÝ 5.5 : (Dạng chuẩn Chomsky, hay CNF)

Một ngôn ngữ phi ngữ cảnh bất kỳ không chứa   đều được sinh ra bằng một văn phạm nào đó mà các luật sinh có dạng A   BC hoặc A   a, với A, B, C là biến còn a là ký hiệu kết thúc.

Chứng minh

Đặt G là CFG sinh ra CFL không chứa . CFG tương đương có dạng chuẩn Chomsky có thể xây dựng từ G theo giải thuật sau :

Bước 1 : Thay thế tất cả các luật sinh có độ dài vế phải bằng 1 (luật sinh đơn vị dạng A   B, với A, B là biến)

Theo định lý 4.4, ta có thể tìm được CFG tương đương G1(V, T, P, S) không có luật sinh đơn vị và luật sinh . Vậy nếu luật sinh mà vế phải chỉ có một ký hiệu thì đó phải là ký hiệu kết thúc và luật sinh này là luật sinh có dạng đúng trong định lý.

Bước 2 : Thay thế các luật sinh có độ dài vế phải >1 và có chứa ký hiệu kết thúc.

Xét luật sinh trong P có dạng A   X1X2 ... Xm (m >1). Nếu Xi là ký hiệu kết thúc a thì ta đưa thêm một biến mới Ca và luật sinh mới Ca   a. Thay thế Xi bởi Ca, gọi tập các biến mới là V' và tập luật sinh mới là P'.

Xét CFG G2 (V', T, P', S). Nếu   G1   thì   *G2   . Vậy L(G1)   L(G2). Ta chứng minh quy nạp theo số bước dẫn xuất rằng nếu A   *G2 w, với A   V và w   T* thì A   *G1 w.

Kết quả hiển nhiên với 1 bước dẫn xuất.

Giả sử kết quả đúng tới k bước dẫn xuất.

Xét A   *G2 w bằng k +1 bước dẫn xuất.

Bước đầu tiên có dạng A   B1B2 ... Bm (m > 1). Ta có thể viết w = w1w2 ...wm trong đó Bi   *G2 wi, 1   i   m. Nếu Bi là ký hiệu kết thúc ai nào đó

thì wi là ai. Theo cách xây dựng P’ ta có luật sinh A   X1X2 ... Xm của P trong đó Xi = Bi nếu Bi V và Xi = ai nếu Bi V’- V. Với Bi   V, ta đã biết rằng có dẫn xuất Bi *G1 wi bằng ít hơn k bước, do vậy theo giả thiết quy nạp Xi *G1 wi. Vậy A *G1 w.

Ta đã có kết quả là một CFL bất kỳ được sinh ra từ một CFG mà mỗi luật sinh có dạng A   a hoặc A   B1B2 ... Bm (m   2) với A, B1, ... ,Bm là các biến và a là ký hiệu kết thúc. Ta sửa G2 bằng cách thêm vào P’ một số luật sinh.

Bước 3 : Thay thế các luật sinh có độ dài vế phải > 2 ký hiệu chưa kết thúc.

Xét luật sinh trong P’có dạng A   B1B2 ... Bm (m > 2) . Ta thay bằng tập hợp các luật sinh : A   B1D1

D1 ® B2D2

...

Dm - 3 ® Bm - 2Dm - 2

Dm - 2 ® Bm - 1Bm

Đặt V’’ là tập các biến mới, P’’ là tập các luật sinh mới và văn phạm mới G3 (V’’, T, P’’, S). Ta có G3 chứa các luật sinh thoả mãn định lý.

Hơn nữa, nếu A *G2   thì A *G3 , vậy L(G2)   L(G3). Ngược lại cũng đúng tức là, L(G3)   L(G2). Chúng ta cũng đã có L(G2)   L(G1) và L(G1)   L(G2). Vậy G3 là văn phạm thoả mãn dạng chuẩn CNF.

Thí dụ 5.12 : Tìm văn phạm có dạng CNF tương đương văn phạm sau :

S ® A | ABA

A ® aA | a | B

B ® bB | b

Bước 1 : Thay thế các luật sinh có độ dài vế phải = 1 (luật sinh đơn vị)

Gọi tập A = {B A * B }, xét các biến trong văn phạm, ta có :

S = { S, A, B }

A = { A, B}

B = { B }

Vậy tập luật sinh mới, theo định lý sẽ chứa các luật sinh không là luật sinh đơn vị trong P, bổ sung thêm các luật sinh mới thay cho luật sinh đơn vị như sau :

S ® aA | a | bB | b | ABA

A ® aA | a | bB | b

B ® bB | b

Bước 2 : Thay thế các luật sinh có độ dài vế phải > 1 và có chứa ký hiệu kết thúc.

Ta thấy, a và b đều xuất hiện ở vế phải một số luật sinh, do đó ta tạo thêm 2 biến mới Ca, Cb và 2 luật sinh mới Ca ® a và Cb ® b.

Văn phạm tương đương có tập luật sinh như sau :

S ® CaA | a | CbB | b | ABA

A ® CaA | a | CbB | b

B ® CbB | b

Ca ® a

Cb ® b

Bước 3 : Thay thế các luật sinh có độ dài vế phải > 2

Chỉ còn duy nhất một luật sinh cần xét ở bước này : S ® ABA và tập luật sinh mới được thay thế có dạng như sau :

S ® CaA | a | CbB| b | AD1

A ® CaA | a | CbB| b

B ® CbB| b

Ca ® a

Cb ® b

D1 ® BA

Cuối cùng, ta sẽ thu được văn phạm có dạng chuẩn Chomsky như trên tương đương với văn phạm đã cho.

Dạng chuẩn Greibach GNF (Greibach Normal Form)

Ta gọi luật sinh với biến A ở bên trái là A - luật sinh.

BỔ ĐỀ 3 : (Dùng thay thế các luật sinh trực tiếp)

Cho G (V, T, P, S) là một CFG, đặt A ® 1B 2 là luật sinh trong P và B ® 1 | 2 | ... | r là các B - luật sinh; văn phạm G1 (V, T, P1, S) thu được từ G bằng cách loại bỏ luật sinh A ® 1B 2 và thêm vào luật sinh A ® 1 1 2 | 1 2 2 | ... | 1 r 2 thì L(G) = L(G1)

Chứng minh

. Hiển nhiên L(G1) L(G) vì nếu A ® 1 i 2 được dùng trong dẫn xuất của G1 thì ta dùng A G 1B 2 G 1 i 2

. Để chỉ ra L(G) L(G1) ta cần chú ý rằng A ® 1B 2 là luật sinh trong P - P1 (có trong G và không có trong G1). Bất cứ khi nào luật sinh A ®

1B 2 được dùng trong dẫn xuất của G thì phải viết lại tại bước sau đó dùng luật sinh dạng B ® i. Hai bước dẫn xuất này có thể được thay thế bằng một bước dẫn xuất duy nhất, hay :

A ® 1B 2 A G1 1 i 2

B ® i

Vậy L(G) = L(G1)

BỔ ĐỀ 4 : (Dùng loại bỏ luật sinh dạng đệ quy trái trong văn phạm)

Đặt G (V, T, P, S) là CFG; A ® A 1 | A 2 | ... | A r là tập các A - luật sinh có A là ký hiệu trái nhất của vế phải (luật sinh đệ quy trái). Đặt A ® 1 | 2 | ... | s là các A - luật sinh còn lại; G1 (V {B}, T, P1, S) là CFG được tạo thành bằng cách thêm biến mới B vào V và thay các A - luật sinh bằng các luật sinh dạng:

1) A ® i

A ® i B với 1 i s

2) B ® i

B ® i B với 1 i r

thì L(G) = L(G1).

Chứng minh

Trong một chuỗi dẫn xuất trái, một chuỗi luật sinh dạng A ® A i phải kết thúc bằng A ® j. Tức là:

A A i1 A i2 i1 ... A ip ip-1… i1 j ip ip-1… i1

Chuỗi dẫn xuất trong G có thể thay bằng chuỗi dẫn xuất trong G1 bởi :

A j BÞ j ipB Þ j ip ip-1…B Þ ... Þ j ip ip-1… i2B

Þ j ip ip-1… i1.

Sự chuyển đổi ngược lại cũng có thể được.

Vậy L(G) = L(G1).

ĐỊNH LÝ 5.6 : (Dạng chuẩn Greibach, hay GNF)

Mỗi CFL bất kỳ không chứa được sinh ra bởi một CFG mà mỗi luật sinh có dạng A a với A là biến, a là một ký hiệu kết thúc, và là một chuỗi các biến (có thể rỗng).

Chứng minh

Bước 1: Đặt G là CFG sinh ra CFL không chứa . Xây dựng văn phạm tương đương G' có dạng chuẩn Chomsky.

Bước 2: Đổi tên các biến trong tập của G' thành A1, A2, ..., Am (m 1) với A1 là ký hiệu bắt đầu. Đặt V = {A1, A2, ..., Am}.

Bước 3: Thay thế các luật sinh sao cho nếu Ai Aj là một luật sinh thì j > i

Bắt đầu từ A1 và tiến tới Am, ta thay thế các Ak - luật sinh :

Nếu Ak Aj là luật sinh với j < k: sinh ra một tập luật sinh mới bằng cách thay thế Aj bên vế phải của mỗi Aj - luật sinh theo bổ đề 3. Lặp lại không quá k - 1 lần ta thu được tập luật sinh dạng Ak Al với l k.

Sau đó, các luật sinh với l = k được thay thế theo bổ đề 4 bằng cách đưa vào các biến mới Bk.

Giải thuật cụ thể như sau:

Begin

(1) For k := 1 to m do begin

(2) for j := 1 to k-1 do

1. for Mỗi luật sinh dạng Ak   Aj do

begin

(4) for Tất cả luật sinh Aj   do

(5) Thêm luật sinh Ak   ;

(6) Loại bỏ luật sinh Ak   Aj

end;

1. for Mỗi luật sinh dạng Ak   Ak do

begin

(8) Thêm các luật sinh Bk   và Bk   Bk;

(9) Loại bỏ luật sinh Ak   Ak

end;

(10) for Mỗi luật sinh Ak   trong đó   không bắt đầu bằng Ak do

(11) Thêm luật sinh Ak   Bk

end;

end;

Bằng cách lặp lại bước xử lý trên cho mỗi biến nguồn, trong P chỉ chứa các luật sinh có dạng như sau :

1) Ai   Aj với j > i

2) Ai   ag với a Î T

3) Bk ® g g Î (V È {B1, B2 , ..., Bi - 1})*

Bước 4: Thay thế các Ai - luật sinh về đúng dạng.

Gọi V’ là tập biến mới phát sinh sau bước 3.

Chú ý rằng ký hiệu trái nhất của vế phải trong một luật sinh bất kỳ đối với biến Am phải là một ký hiệu kết thúc, vì Am là biến có chỉ số cao nhất. Ký hiệu trái nhất của vế phải của một Am-1- luật sinh bất kỳ phải là Am hoặc một ký hiệu kết thúc. Nếu là Am, ta tạo ra tập luật sinh mới bằng cách thay thế Am bởi chuỗi vế phải của các Am-luật sinh theo bổ đề 3. Tiếp tục quá trình này cho các luật sinh từ Am-2, ..., A2, A1 cho tới khi vế phải của tất cả các Ai - luật sinh có dạng bắt đầu bằng một ký hiệu kết thúc.

Bước 5: Thay thế các Bk -luật sinh về đúng dạng.

Bước cuối cùng, ta khảo sát các luật sinh với tập các biến mới B1, B2, ..., Bm.

Vì ta bắt đầu từ văn phạm đã có dạng chuẩn Chomsky, nên dễ dàng chứng minh quy nạp theo số lần áp dụng bổ đề 3 và bổ đề 4 rằng vế phải của mỗi Ai -luật sinh, với 1   i   n, bắt đầu bằng ký hiệu kết thúc hoặc AjAk với j, k nào đó. Vậy   (trong bước (7)) không khi nào có thể rỗng hoặc bắt đầu bằng một Bj khác, hay tất cả Bi - luật sinh đều có vế phải bắt đầu bằng ký hiệu kết thúc hoặc Ai. Một lần nữa, lại áp dụng bổ đề 3 cho mỗi Bi - luật sinh.

Ta thu được tập luật sinh trong văn phạm sau cùng thỏa đúng dạng chuẩn Greibach.

Thí dụ 5.13 : Tìm văn phạm có dạng GNF tương đương văn phạm G sau :

A1 ® A2A1 | A2A3

A2 ® A3A1 | a

A3 ® A2A2 | b

Bước 1 : G thỏa dạng chuẩn CNF sinh ra CFL không chứa

Bước 2 : Ta có V = {A1, A2, ..., A3}

Bước 3 : Thay thế các luật sinh sao cho nếu Ai   Aj   là một luật sinh thì j > i.

Ta thấy trong tập luật sinh, các luật sinh cho A1 và A2 đã thỏa điều kiện j > i. Chỉ có luật sinh A3 ® A2A2 cần sửa đổi. Áp dụng bổ đề 3 để thay thế luật sinh này, ta có:A3 ® A3A1A2 | aA2

Sau đó, dùng bổ đề 4 để loại bỏ đệ quy trái, ta được tập luật sinh mới có dạng như sau :

A1 ® A2A1 | A2A3

A2 ® A3A1 | a

A3 ® aA2 | b | aA2B | bB

B ® A1A2 | A1A2 B

Bước 4 : Thay thế các Ai -luật sinh về đúng dạng.

Ở bước này, ta có thể thấy tất cả các A3 - luật sinh đã có dạng chuẩn. Tiếp tục, áp dụng bổ đề 3 để thay thế các A3 - luật sinh vào A2, A1, thu được tập luật sinh mới như sau:

A1 ® aA2A1A1 | bA1A1 | aA2BA1A1 | bBA1A1 | aA1|

aA2A1A3 | bA1A3 | aA2BA1A3 | bBA1A3 | aA3

A2 ® aA2A1 | bA1 | aA2BA1| bBA1| a

A3 ® aA2 | b | aA2B | bB

B ® A1A2 | A1A2 B

Bước 5 : Thay thế các Bk - luật sinh về đúng dạng.

B ® aA2A1A1A2 | bA1A1A2 | aA2BA1A1A2 | bBA1A1A2 | aA1A2

| aA2A1A3A2 | bA1A3A2 | aA2BA1A3A2 | bBA1A3A2 | aA3A2

| aA2A1A1A2B | bA1A1A2B | aA2BA1A1A2B | bBA1A1A2B | aA1A2 B

| aA2A1A3A2B | bA1A3A2B | aA2BA1A3A2B | bBA1A3A2B | aA3A2B

Cuối cùng, ta thu được văn phạm có dạng GNF với 39 luật sinh.

TÍNH CHẤT CỦA NGÔN NGỮ PHI NGỮ CẢNH

Cũng như lớp ngôn ngữ chính quy, có một vài tính chất giúp xác định một ngôn ngữ có thuộc lớp ngôn ngữ phi ngữ cảnh hay không ?

Bổ đề bơm đối với CFL

(Dùng chứng minh một ngôn ngữ không là ngôn ngữ phi ngữ cảnh)

Cho L là một CFL bất kỳ, tồn tại một số n chỉ phụ thuộc vào L sao cho nếu z   L và | z | n thì ta có thể viết z = uvwxy sao cho:

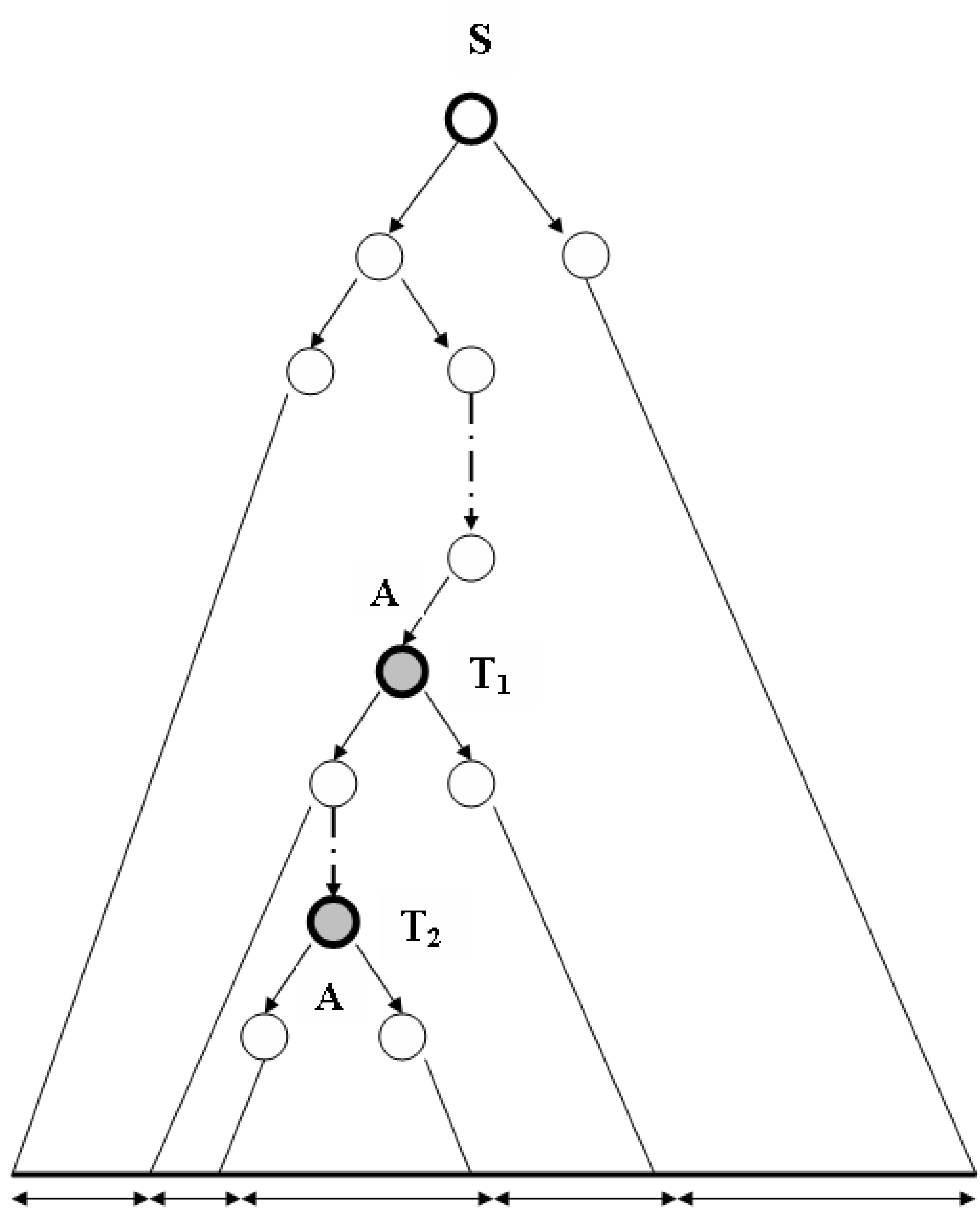
1) | vx | ³ 1

2) | vwx | n và

3)   i ³ 0 : uviwxiy Î L

Chứng minh

Đặt G là văn phạm có dạng chuẩn CHOMSKY sinh L - { }. Chú ý rằng nếu z   L(G) và cây dẫn xuất không có đường đi dài hơn i thì chuỗi sinh ra từ văn phạm có độ dài không dài hơn 2 i -1.



z3 z2 z4

z = u v w x y

Hình 5.5 - Các bước dẫn xuất trong chứng minh Bổ đề bơm

Giả sử G có k biến, ta đặt n = 2k. Nếu z   L(G) và | z | n thì | z | > 2k-1, vậy có một đường đi nào đó trên cây dẫn xuất có độ dài lớn hơn hoặc

bằng k+1. Như vậy đường đi đó sẽ có ít nhất k+2 đỉnh, hay có ít nhất k+1 biến trên đường đi (chỉ có nút lá mới có thể không là biến), suy ra phải có biến xuất hiện hai lần, hơn nữa ta phải có:

1) Có hai đỉnh v1 và v2 có cùng nhãn là A

2) Đỉnh v1 gần gốc hơn v2

3) Phần đường đi từ v1 tới lá có độ dài nhiều nhất là k+1 (đi từ lá lên tới gốc theo đường đi, chỉ có lá mới có thể là ký hiệu kết thúc vì vậy trong k+2 đỉnh đầu tiên phải có ít nhất k+1 biến và phải có ít nhất hai biến trùng nhau)

Xét cây con T1 có đỉnh là v1 biểu diễn dẫn xuất của chuỗi con có độ dài không quá 2k (vì trong cây con T1 không có đường đi nào có độ dài vượt quá k+1). Đặt z1 là chuỗi sinh ra từ cây T1. Ta gọi T2 là một cây con có nút gốc là v2, rõ ràng T2 là cây con của T1. Giả sử T2 sinh ra chuỗi z2 thì ta có thể viết z1 = z3z2z4. Hơn nữa z3 và z4 không thể đồng thời bằng vì luật sinh đầu tiên trong cây dẫn xuất của T1 là A   BC với biến B, C nào đó. Cây con T2 phải thuộc vào cây con sinh bởi bút biến B hoặc cây con sinh bởi nút biến C. Ta có :

A *G z3Az4 và A *G z2 trong đó | z3z2z4 | 2k = n.

Vậy A *G z3i z2 z4i , i   0.

Hiển nhiên chuỗi z = uz3z2z4y, với các chuỗi u, y nào đó.

Nếu đặt z3 = v, z2 = w và z4 = x, thì ta sẽ hoàn thành việc chứng minh.

Ứng dụng bổ đề bơm

Thí dụ 5.14 :Chứng minh L = {aibici | i   1} không phải là ngôn ngữ phi ngữ cảnh.

Chứng minh

Giả sử L là ngôn ngữ phi ngữ cảnh, khi đó có tồn tại số n (theo bổ đề bơm).

Xét chuỗi z = anbncn với | z | n, ta có thể viết z = uvwxy thoả mãn bổ đề.

Ta thấy vx nằm trong anbncn và | vwx | n, vậy vx không thể chứa cả ký hiệu a và ký hiệu c (do sau ký hiệu a bên phải nhất n+1 vị trí mới đến vị trí của c bên trái nhất). Nếu vx chỉ có chứa ký hiệu a, thì chuỗi uwy (trường hợp uviwxiy với i = 0) sẽ có chứa số ký hiệu b và c ít hơn số ký hiệu a vì | vx | 1. Vậy uwy không có dạng ajbjcj. Tương tự cho các trường hợp chuỗi vx chỉ chứa ký hiệu b hay c. Còn nếu trong vx có chứa ký hiệu a và b thì chuỗi uvy sẽ có chứa số ký hiệu c lớn hơn a và b, nên nó cũng không thể thuộc L. Cũng tương tự cho trường hợp vx chứa hai ký hiệu b và c. Cuối cùng, ta suy ra chuỗi uviwxiy   L, vì các ký hiệu a, b, c trong chúng không thể bằng nhau với mọi i. Mà theo giả thiết của bổ đề bơm, chuỗi này phải thuộc L, mâu thuẫn.

Vậy L không thể là CFL.

Thí dụ 5.15 :Chứng minh L = {aibjcidj | i, j   1} không phải là ngôn ngữ phi ngữ cảnh.

Chứng minh

Giả sử L là ngôn ngữ phi ngữ cảnh, khi đó có tồn tại số n (theo bổ đề bơm).

Xét chuỗi z = anbncndn với | z | n, ta có thể viết z = uvwxy thoả mãn bổ đề.

Ta thấy vì vx nằm trong anbncndn và | vwx | n, nên vx không thể chứa ít nhất hai ký hiệu khác nhau. Hơn nữa, nếu vx có chứa hai ký hiệu khác nhau, thì chúng phải là hai ký hiệu liên tiếp đứng cạnh nhau, chẳng hạn a và b. Nếu vx chỉ có chứa ký hiệu a, thì chuỗi uwy sẽ có số ký hiệu a ít hơn số ký hiệu c nên không thuộc L, mâu thuẫn. Tương tự với trường hợp chuỗi vx chỉ chứa ký hiệu b, c hoặc d. Bây giờ giả sử chuỗi vx có

chứa a và b thì vwy vẫn có số ký hiệu a ít hơn c. Mâu thuẫn tương tự cũng xuất hiện khi chuỗi vx có chứa b và c hoặc c và d. Vì chỉ có thể có một trong các trường hợp này nên ta có thể kết luận rằng L không thể là CFL.

Câu hỏi :

?

Hãy so sánh các yếu tố ràng buộc trong phát biểu Bổ đề bơm cho ngôn ngữ phi ngữ cảnh và Bổ đề bơm cho ngôn ngữ chính quy ?

Tính chất đóng của CFL

ĐỊNH LÝ 5.7 : CFL đóng với phép hợp, phép nối kết và phép bao đóng Kleene.

Chứng minh

Đặt L1 và L2 là hai CFL sinh bởi các CFG G1 (V1, T1, P1, S1) và G2 (V2, T2, P2, S2).

Vì các biến có thể đổi tên mà không ảnh hưởng tới ngôn ngữ sinh ra nên ta có thể xem tập V1 và V2 là rời nhau. Ta cũng giả sử các biến mới S3, S4, S5   V1 hoặc V2

. Đối với L1   L2: Xây dựng văn phạm G3 (V1   V2   {S3}, T1   T2, P3, S3),

trong đó P3 = P1   P2   {S3   S1 | S2}.

Nếu w   L1 thì dẫn xuất S3 G3 S1 *G1 w là một dẫn xuất trong G3 (vì mỗi luật sinh trong G1 cũng là luật sinh trong G3). Tương tự mỗi chuỗi trong L2 có dẫn xuất trong G3 bắt đầu bằng S3   S2. Vậy L1   L2 L(G3).

Ngược lại, nếu w   L(G3) thì dẫn xuất S3 *G3 w phải bắt đầu bằng S3   S1 hoặc S3   S2. Tức là dẫn xuất có dạng S3 G3 S1 *G3 w hoặc S3 G3 S2 *G3 w. Trong trường hợp thứ nhất, do V1 và V2 rời nhau nên chỉ có các ký hiệu của G1 xuất hiện trong dẫn xuất S1 *G3 w. Vì trong các luật sinh của P3 chỉ có chứa các ký hiệu thuộc G1 và nằm trong tập luật sinh P1, nên ta có thể kết luận chỉ có những luật sinh thuộc P1 được dùng trong dẫn xuất S1 *G3 w. Vì thế S1 *G1 w và w   L1. Tương tự cho trường hợp dẫn xuất S3 G3 S2, ta cũng có w   L2. Vậy L(G3)   L1   L2, và vì thế L(G3) = L1   L2.

Vậy ta đã chứng minh xong L(G3) = L1   L2, hay L1   L2 là CFL.

. Đối với L1L2 : Xây dựng văn phạm G4 (V1   V2   {S4}, T1   T2, P4, S4) ,

trong đó P4 = P1   P2   {S4   S1S2}.

Chứng minh tương tự như trên ta có L(G4) = L1L2, vậy L1L2 cũng là CFL.

. Đối với L1* : Xây dựng văn phạm G5 (V1   {S5}, T1, P5, S5),

trong đó P5 = P1   { S5   S1S5 | }.

Ta cũng dễ dàng chứng minh được L(G5) = (L(G1))*.

ĐỊNH LÝ 5.8 : CFL không đóng với phép giao

Chứng minh

Ta đã biết ngôn ngữ L1 = {aibici | i   1} không là CFL. Ta có thể chứng minh :

. L2 = {aibicj | i   1 và j   1} là CFL vì L2 được sinh bởi văn phạm :

S ® AB

A ® aAb | ab

B ® cB | c

. L3 = {aibjcj | i   1 và j   1} cũng là CFL vì L3 được sinh từ văn phạm :

S ® CD

C ® aC | a

D ® bDc | bc

Tuy nhiên L2   L3 = L1 không phải là CFL.

Vậy CFL không đóng với phép giao.

Hệ quả: CFL không đóng với phép lấy phần bù.

Chứng minh

Giả sử CFL đóng với phép lấy phần bù, vậy với L1, L2 là hai CFL bất kỳ, theo quy luật DeMorgan ta có L_1 L_2 = L_1 L_2 nên L1   L2 là CFL hay CFL cũng đóng với phép giao. (Điều này mâu thuẫn với định lý 6.6)

Câu hỏi :

?

Hãy so sánh các tính chất đóng của lớp ngôn ngữ phi ngữ cảnh với lớp ngôn ngữ chính quy ?

CÁC GIẢI THUẬT QUYẾT ĐỊNH CFL

Có một vài câu hỏi về CFL mà chúng ta cần phải trả lời. Chẳng hạn, liệu một ngôn ngữ phi ngữ cảnh cho trước là rỗng, hữu hạn hay vô hạn hay một chuỗi nào đó liệu có thuộc ngôn ngữ này không ? Tuy nhiên, cũng có những câu hỏi về CFL mà không có giải thuật nào để có thể trả lời. Chẳng hạn, liệu hai CFG thì có tương đương nhau, hay phần bù của một CFL có là CFL hay không, hoặc một CFG cho trước nào đó có phải

là văn phạm mơ hồ ? Trong phần này, chúng ta chỉ đưa ra giải thuật cho một số các câu hỏi có thể trả lời.

Giải thuật xác định ngôn ngữ phi ngữ cảnh

ĐỊNH LÝ 5.9 : Tồn tại giải thuật để xác định CFL là: rỗng, hữu hạn, vô hạn.

Chứng minh

Với văn phạm G (V, T, P, S) :

. Để kiểm tra L(G) có rỗng hay không, ta dùng bổ đề 5. 1: Rõ ràng L(G) không rỗng khi và chỉ khi S sinh ra một chuỗi ký hiệu kết thúc nào đó.

. Để kiểm tra L(G) hữu hạn hay vô hạn, ta dùng định lý 5. 5 để tìm văn phạm tương đương G' (V', T, P', S) có dạng chuẩn CHOMSKY và không có ký hiệu vô ích sinh ra L(G) - { }. L(G) hữu hạn khi và chỉ khi L(G') hữu hạn.

Để kiểm tra tính hữu hạn của CFG có dạng chuẩn CHOMSKY, ta chỉ cần vẽ đồ thị có hướng với mỗi đỉnh trên đồ thị là một biến thuộc văn phạm và cạnh từ A đến B nếu và chỉ nếu có luật sinh A BC hoặc A CB với biến C bất kỳ. Khi đó, ngôn ngữ sinh ra là hữu hạn nếu và chỉ nếu đồ thị không có chu trình. Vì :

Nếu đồ thị có chu trình, giả sử chu trình là A0, A1,... , An, A0 thì sẽ có chuỗi dẫn xuất: A0 1A1 1 2A2 2 ... nAn n n+1 A0 n+1, trong đó i, i là chuỗi các biến và | i i | = i. Vì không có ký hiệu vô ích nên n+1 * w và n+1 * x với mọi chuỗi w, x là các chuỗi ký hiệu kết thúc và độ dài tổng cộng ít nhất bằng n+1. Vì n 0, nên w và x không thể đồng thời bằng .

Kế tiếp, cũng do văn phạm không có chứa ký hiệu vô ích nên ta có thể tìm được các chuỗi y, z sao cho S * yA0z và chuỗi ký hiệu kết thúc v sao

cho A0   * v. Vậy   i ta có :

S   * yA0z   * ywA0xz   * yw2A0x2z   * ...   * ywiA0xiz   * ywivwiz.

Vì | wx | > 0, nên chuỗi ywivwiz không thể bằng ywjvwjz nếu i   j. Vậy văn phạm sinh ngôn ngữ vô hạn.

Ngược lại, giả sử đồ thị không có chu trình. Ta gọi hạng của biến A là độ dài lớn nhất của đường đi bắt đầu từ A. Vì không có chu trình nên A sẽ có hạng hữu hạn. Nếu A   BC là một luật sinh thì hạng của B và C phải nhỏ hơn hạng của A. Ta chứng minh quy nạp theo r (hạng của A) rằng không có chuỗi ký hiệu kết thúc nào có độ dài lớn hơn 2r

Với r = 0: hạng của A bằng 0, vậy không có cạnh từ A. Do đó, tất cả các A - luật sinh đều có dạng A   a, hay A dẫn ra chuỗi có độ dài l = 20.

Xét r > 0: nếu ta dùng luật sinh A   a thì dẫn ra chuỗi chỉ có độ dài 1, nếu dùng luật sinh A   BC thì vì B, C có hạng í hơn hoặc bằng r -1 nên theo giả thiết quy nạp B, C dẫn ra chuỗi có độ dài ngắn hơn 2r -1 . Vậy BC không thể dẫn ra chuỗi có độ dài lớn hơn 2r. Giả sử S có hạng là r0 thì các chuỗi do S sinh ra có độ dài không quá 2r0 . Vì thế suy ra ngôn ngữ là hữu hạn.

Thí dụ 5.16 : Xét văn phạm G chứa các luật sinh sau :

S ® AB

A ® BC | a

B ® CC | b

C ® a

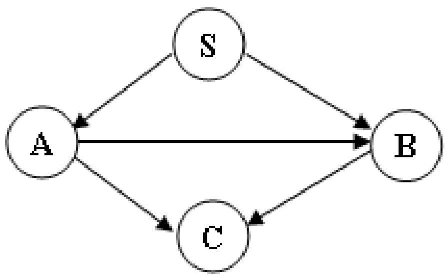
Ta thấy văn phạm G có các luật sinh đã thỏa dạng chuẩn Chomsky.

. Để kiểm tra tính rỗng của văn phạm, ta áp dụng Bổ đề 5.1 lên tập biến V để tìm tập biến mới mới V1 chỉ chứa các biến có khả năng dẫn ra chuỗi ký hiệu kết thúc trong văn phạm :

Ta có : V1 = { A, B, C, S } vì A ® a , B ® b, C ® a và S ® AB

Hay S   V1 có nghĩa là S có thể sinh ra các chuỗi ký hiệu kết thúc. Vậy ngôn ngữ sinh bởi văn phạm G : L(G) không rỗng.

. Để kiểm tra tính hữu hạn của văn phạm, ta vẽ đồ thị có hướng tương ứng với các luật sinh trong văn phạm như sau :



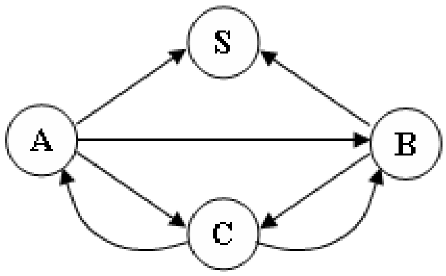
Hình 5.6 - Đồ thị có hướng tương ứng

Rõ ràng, ta thấy đồ thị không có chu trình. Hạng của S, A, B, C lần lượt là 3, 2, 1 và 0. Chẳng hạn, một đường đi dài nhất từ S là S   A   B   C. Vậy văn phạm này là hữu hạn, nó sinh ra hữu hạn chuỗi và độ dài các chuỗi không lớn hơn 23 = 8.

Thực tế, chuỗi dài nhất dẫn xuất được từ S là :

S   AB   BCB   CCCB   CCCCC   * aaaaa ,với độ dài chuỗi là 5.

Nếu ta thêm vào văn phạm một luật sinh mới : C   AB, thì đồ thị có hướng tương ứng lúc đó có dạng như sau :



Hình 5.7 - Đồ thị có hướng tương ứng văn phạm bổ sung

Đồ thị mới này có nhiều chu trình, chẳng hạn A B C A. Vậy ta phải tìm được một dẫn xuất dạng A * 3A 3 , cụ thể là A BC CCC CABC, trong đó 3 = C và 3 = BC. Vì C * a và BC * ba nên A * aAba.

Mặt khác, S * Ab và A * a, suy ra : S * aia(ba)ib, i. Vậy ngôn ngữ sinh từ văn phạm mới là vô hạn.

Giải thuật thành viên (Membership)

ĐỊNH LÝ 5.10 : Tồn tại giải thuật để xác định với một CFL nào đó sinh ra từ CFG G(V, T, P, S) và một chuỗi x bất kỳ thì x có thuộc L(G) hay không ?

Chứng minh

Có một vài giải thuật được đề nghị cho bài toán thành viên này. Sau đây trình bày một giải thuật theo vòng lặp đơn giản, ta gọi là giải thuật CYK (Cocke-Younger-Kasami) với thời gian tỷ lệ với x 3.

Giả sử văn phạm G (V, T, P, S) đã có dạng chuẩn Chomsky và x = n 1. Trước hết, ta phải xác định với mỗi i, j và mỗi biến A, phải chăng A * xij , trong đó xij là một chuỗi con của chuỗi x tính từ vị trí thứ i và có độ dài j.

Ta chứng minh quy nạp theo độ dài j :

- Với j = i : ta có A * xij khi và chỉ khi A xij là một luật sinh.

- Với j > i : ta có A * xij khi và chỉ khi có một luật sinh dạng A BC và số k, 1 k < j sao cho B dẫn xuất ra k ký hiệu đầu tiên của xij và C dẫn xuất ra j – k ký hiệu cuối của xij. Có nghĩa là :

B * xik và C * x i+ k, j - k

Vì cả k và j – k đều nhỏ hơn j, nên theo giả thiết quy nạp ta đã xác định được liệu hai chuỗi dẫn xuất này có tồn tại hay không ? Thế thì cũng có thể c]xác định được liệu A * xij hay không ?

Với cách thực hiện như thế, ta sẽ xác định được phải chăng S * x1n , nhưng x1n = x, vậy x L(G) khi và chỉ khi S * x1n.

Sau đây trình bày giải thuật CYK theo giải thuật trên, trong đó Vij là tập hợp tất cả các biến A sao cho A * xij . Chú ý rằng ta có thể giả thiết 1 i n – j + 1, bởi vì lúc đó chuỗi con xij với độ dài j mới thực sự tồn tại.

Bước (1) và (2) xử lý trường hợp j = i. Vì văn phạm G đã cho sẵn, cho nên bước (2) chiếm mộ thời gian cố định. Vậy các bước (1) và (2) chiếm thời gian O(n). các vòng lặp For ở các dòng (3) và (4) làm cho các bước từ (5) đến (7) lặp lại nhiều nhất là n2 lần (do i, j n). Bước (5) mỗi lần thực hiện cũng chiếm một khoảng thời gian cố định. Vậy tổng thời gian để thực hiện bước (5) là O(n2). Vòng lặp For ở dòng (6) làm cho bước (7) lặp lại n lần hoặc ít hơn. Vì bước (7) cũng chiếm một thời gian cố định, nên các bước (6) và (7) gộp lại chiếm thời gian O(n). Vì các bước này được thực hiện O(n2), nên tổng thời gia thực hiện cho bước (7) là O(n3). Vậy thời gian thực hiện toàn bộ giải thuật là ở cấp O(n3).

Giải thuật CYK:

Begin

(1) For i := 1 to n do

(2)Vij := { A A a là một luật sinh và a là ký hiệu thứ i trong x }

(3) For j := 2to n do

(4)for i := 1 to n – j + 1 do

begin

(5)Vij := 0;

(6)for k := 1 to j - 1 do

(7) Vij := Vij { A A BC là một luật sinh, B Vik

và C Vi+ k, j – k }

end;

End;

Thí dụ 5.17 : Cho văn phạm G có dạng chuẩn Chomsky chứa các luật sinh sau :

S ® AB | BC

A ® BA | a

B ® CC | b

C ® AB | a

Xét chuỗi nhập x = baaba.

Bảng các Vij cho ở hình 5.8 dưới đây. Dòng đầu tiên trong bảng được cho bởi các bước (1) và (2) trong giải thuật. Vì x11 = x41 = b nên V11 = V41 = {B} vì B là biến duy nhất dẫn xuất ra b, còn x21 = x31 = x51 = a, suy ra V21 = V31 = V51 = {A, C} vì A và C có các luật sinh với a bên vế phải.

	b	a	a	b	a
		i			
j	1	2	3	4	5
1	B	A, C	A, C	B	A, C
2	S, A	B	S, C	S, A	
3		B	B		
4		S, A, C			
5	S, A, C				

Hình 5.8 – Bảng các Vij

Để tính Vij với j > i, ta phải thực hiện vòng lặp For ở bước (6) và (7). Ta phải đối chiếu Vik với Vi+ k, j – k với k = 1, 2, …, j - 1 để tìm biến D trong Vik và biến E trong Vi+ k, j – k sao cho DE là vế phải của một hay nhiều luật sinh. Các vế trái của những luật sinh đó được đưa vào trong Vij. Quá trình đối chiếu đó diễn ra bằng cách giảm dần giá trị cột i, đồng thời tăng dần lên theo đường chéo qua Vij về phía phải như các chiều mũi tên vẽ trong hình 5.9.

	Vij				

Hình 5.9 – Quá trình tính các Vij

Chẳng hạn, để tính V24, đầu tiên ta đối chiếu V21 = {A, C} với V33 = {B}. Ta có V21 V33 = {AB, CB}. Vì có các luật sinh S AB và C AB nên S và C được đưa vào V24. Tiếp đến ta lại xét V22 V42 = {A}{S, A} = {BS, BA}. Vì có luật sinh A BA, vậy ta đưa thêm A vào V24. Cuối cùng ta xét V23 V51 = {B}{A, C} = {BA, BC} gặp lại các vế phải đã xét, vậy không thể thêm gì vào V24. Vậy V24 = {S, AC}.

Cuối cùng, vì S V15 , vậy chuỗi baaba là thuộc ngôn ngữ sinh ra bởi văn phạm đã cho.

Tổng kết chương V: Việc mô tả ngôn ngữ phi ngữ cảnh bằng phương tiện văn phạm phi ngữ cảnh tỏ ra rất hữu hiệu, cũng tương tự như việc sử dụng văn phạm BNF trong định nghĩa các ngôn ngữ lập trình. Trong chương này, chúng ta đã khảo sát tương đối cặn kẽ các phương tiện mô tả ngôn ngữ của văn phạm CFG thông qua các giải thuật tối ưu biến, giản lược, quy chuẩn và các tính chất của lớp ngôn ngữ mà nó mô tả. Câu hỏi đặt ra tiếp theo là có hay không một lớp ôtômát tương ứng để nhận dạng lớp ngôn ngữ phi ngữ cảnh. Như chúng ta đã thấy, lớp ngôn ngữ phi ngữ cảnh thực sự chứa lớp ngôn ngữ chính quy trong đó, nên ôtômát hữu hạn không thể nhận biết tất cả ngôn ngữ phi ngữ cảnh. Một cách trực quan, ôtômát hữu hạn có bộ nhớ bị hạn chế nghiêm ngặt, trong khi việc nhận dạng CFL có thể yêu cầu phải lưu trữ một lượng thông tin khá lớn. Khả năng cho sự mở rộng này sẽ được chúng ta xét đến trong nội dung của chương tiếp theo.

BÀI TẬP CHƯƠNG V

- . Hãy mô tả ngôn ngữ sinh bởi các văn phạm sau :

a) S ® aS | Sb | aSb | c

b) S ® SS | a | b

c) S ® SaS | b

d) S ® aSS | b

e)S ® aA | bB| c

A ® Sa

B ® Sb

f)S ® AB

A ® Sc | a

B ® dB | b

g)S ® TT

T ® aTa | bTb | c

5.2. Hãy chỉ ra một văn phạm phi ngữ cảnh sinh ra tập hợp :

a) Tập hợp các chuỗi đọc xuôi đọc ngược như nhau trên bộ chữ cái = {0,1}.

b) Tập hợp chuỗi các dấu ngoặc đúng trong biểu thức số học.

c) Tập hợp {aibicj i, j 0}

d) Tập hợp {ambn m, n > 0}

e) Tập hợp {aicaj i j 0}

f) Tập hợp {ajbjcidi i, j 1}

5.3. Cho văn phạm G với các luật sinh sau :

S ® aB | bA

A ® a | aS | bAA

B ® b | bS | aBB

Với chuỗi aaabbabbba , hãy tìm:

a) Dẫn xuất trái nhất.

b) Dẫn xuất phải nhất.

c) Cây dẫn xuất.

d) Văn phạm này có là văn phạm mơ hồ không ?

5.4. Cho văn phạm G với các luật sinh sau :

E ® T | E + T | E - T

T ® F | T   F | T / F

F ® a | b | c | (E)

Hãy vẽ cây dẫn xuất sinh ra các chuỗi nhập sau :

a) a – (b   c / a)

b) a   (b - c)

c) (a + b) / c

5.5. Cho văn phạm : S ® aSbS | bSaS |

a) Chứng tỏ văn phạm này là văn phạm mơ hồ .

b) Xây dựng dẫn xuất trái (phải) và cây dẫn xuất tương ứng cho chuỗi abab.

c) Văn phạm này sinh ra ngôn ngữ gì ?

5.6. Chứng tỏ văn phạm sau đây là mơ hồ :

S ® If b then S else S

S ® If b then S

S ® a

Trong đó a, b, if, then, else là các ký hiệu kết thúc và S là biến.

5.7. Chứng tỏ văn phạm sau đây là mơ hồ :

S ® aBS | aB | bAS | bA

A ® bAA | a

B ® aBB | b

Hãy đề nghị một văn phạm không mơ hồ tương đương ?

5.8. Tìm CFG không có chứa ký hiệu vô ích tương đương với văn phạm:

a)S ® A | a

A ® AB

B ® b

b)S ® AB | CA

A ® a

B ® BC | AB

C ® aB | b

5.9. Tìm văn phạm tương đương với văn phạm sau không có chứa ký hiệu vô ích, luật sinh   và luật sinh đơn vị :

a)S ® aSbS | bSaS |

b) S ® A | B

A ® aB | bS | b

B ® AB | Ba

C ® AS | b

c) S ® ABC

A ® BB |

B ® CC | a

C ® AA | b

d) S ® A | B

A ® C | D

B ® D | E

C ® S | a |

D ® S | b

E ® S | c |

5.10. Tìm văn phạm chỉ có chứa một luật sinh duy nhất S ® tương đương với văn phạm sau :

S ® AB

A ® SA | BB | bB

B ® b | aA |

5.11. Biến đổi các văn phạm sau đây về dạng chuẩn CHOMSKY:

a)S ® bA | aB

A ® bAA | aS | a

B ® aBB | bS | b

b)S ® aAB | BA

A ® BBB| a

B ® AS| b

c)S ® adAda | aSa | aca

A ® bAb | bdSdb

d) S ® 0S1 | 01

e) S ® #S | [S   S] | p | q

5.12. Biến đổi các văn phạm sau đây về dạng chuẩn GREIBACH:

a) G ({S, A}, {0, 1}, P, S) với các luật sinh :

S ® AA | 0

A ® SS | 1

b) G ({A1, A2, A3}, {a, b}, P, A1) với các luật sinh :

A1 ® A2A3

A2 ® A3A1 | b

A3 ® A1A2 | a

c) G ({A1, A2, A3, A4}, {a, b}, P, A1) với các luật sinh :

A1 ® A2A3 | A3A4

A2 ® A3A2 | a

A3 ® A4A4 | b

A4 ® A2A3 | a

5.13. Chứng minh rằng các ngôn ngữ sau không phải là CFL:

a)L = {ai bj ck i < j < k }

b)L = {ai bj j = i2 }

c)L = {ai i là số nguyên tố }

d) L = {anbncndn n 0}

BÀI TẬP LẬP TRÌNH

5.14. Viết chương trình loại bỏ các ký hiệu vô ích trong một CFG.

5.15. Viết chương trình chuẩn hóa một CFG thành dạng chuẩn CHOMSKY (CNF).

5.16. Viết chương trình chuẩn hóa một CFG thành dạng chuẩn GREIBACH (GNF).

Ôtômát đẩy xuống
Nội dung chính: Trong chương này, chúng ta khảo sát một dạng mô hình ôtômát khác, có khả năng nhận diện được lớp ngôn ngữ mà văn phạm phi ngữ cảnh sinh ra - ôtômát đẩy xuống (PDA) - với sự bổ sung thêm của Stack đóng vai trò như một bộ giữ nhớ trong quá trình ôtômát thực hiện các phép chuyển để nhận dạng ngôn ngữ. Tiếp theo đó, mối quan hệ tương đương giữa hai cơ chế - ôtômát đẩy xuống và CFG- dùng biểu diễn cho lớp văn phạm phi ngữ cảnh cũng sẽ được nêu ra và chứng minh chặt chẽ. Mục tiêu cần đạt : Cuối chương này, sinh viên có thể: Thiết kế PDA chấp nhận một ngôn ngữ phi ngữ cảnh cho trước bằng Stack rỗng hay trạng thái kết thúc. Chuyển một PDA chấp nhận ngôn ngữ bằng trạng thái kết thúc sang PDA chấp nhận ngôn ngữ bằng Stack rỗng và ngược lại. Xây dựng NPDA chấp nhận ngôn ngữ sinh ra từ một CFG. Viết văn phạm phi ngữ cảnh sinh ra lớp ngôn ngữ được chấp nhận bởi một NPDA cho trước. Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, sinh viên cần nắm vững các tính chất của lớp ngôn ngữ phi ngữ cảnh; cơ chế đoán nhận ngôn ngữ của dạng máy trừu tượng ôtômát và các thành phần bắt buộc của chúng; … Tài liệu tham khảo : [1] V.J. Rayward-Smith – A First course in Formal Language Theory (Second Editor) – McGraw-Hill Book Company Europe – 1995 (Chapter 6 : Pushdown Automata) [2] John E. Hopcroft, Jeffrey D.Ullman – Introduction to Automata Theory, Languages and Computation – Addison – Wesley Publishing Company, Inc – 1979 (Chapter 5 : Pushdown Automata) [3] Hồ Văn Quân – Giáo trình lý thuyết ôtômát và ngôn ngữ hình thức – Nhà xuất bản Đại học quốc gia Tp. Hồ Chí Minh – 2002. [4] Copy right by David Matuszek - NPDAs and CFGs: http://www.netaxs.com/people/nerp/automata/npda-cfg0.html

ÔTÔMÁT ĐẨY XUỐNG (PDA : PUSHDOWN AUTOMATA)

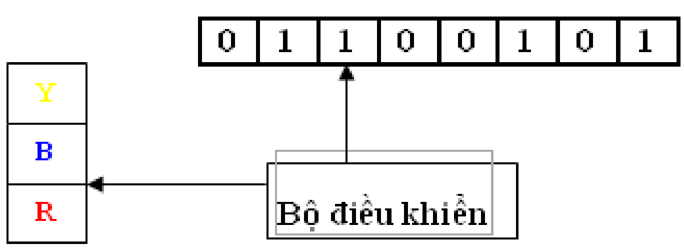
Như ta đã biết, lớp các ngôn ngữ chính quy được sinh từ văn phạm chính quy, đồng thời cũng được đoán nhận bởi các ôtômát hữu hạn (đơn định hoặc không đơn định). Trong phần này, chúng ta lại thấy một điều tương tự là lớp ngôn ngữ phi ngữ cảnh, được sinh ra từ văn phạm phi ngữ cảnh, cũng được đoán nhận bởi một loại ôtômát khác - gọi là ôtômát đẩy xuống (PDA).

Có một điều khác biệt là ở đây, chỉ có dạng ôtômát đẩy xuống không đơn định (NPDA) mới có thể đủ mạnh để đoán nhận lớp ngôn ngữ phi ngữ cảnh, còn dạng đơn định (DPDA) chỉ cho phép đoán nhận một tập con thực sự của lớp ngôn ngữ này. Tuy nhiên, tập con đó cũng bao gồm phần lớn các ngôn ngữ lập trình.

Mô tả PDA

Ôtômát đẩy xuống thực chất là một ôtômát với sự điều khiển cả hai: băng nhập và Stack (bộ đẩy xuống). Về cơ bản, nó vẫn giữ tất cả các thành phần của một ôtômát hữu hạn, với sự bổ sung thêm một ngăn xếp làm việc (Stack) đóng vai trò như một bộ giữ nhớ, nhờ đó mà khả năng ghi nhớ của ôtômát được tăng thêm. Stack xem như là một chồng đĩa, vì vậy cái đặt vào sau sẽ lấy ra trước (LIFO).

Với sự mở rộng này ôtômát đẩy xuống có thể chấp nhận cả các biểu thức không chính quy. Chẳng hạn tập hợp L = {wcwR | w   (0+1)*} (với wR là chuỗi đảo ngược của chuỗi w) là một ngôn ngữ phi ngữ cảnh sinh bởi văn phạm S   0S0 | 1S1 | c và nó không thể được chấp nhận bởi bất kỳ một ôtômát hữu hạn nào.



Hình 6.1 - Mô tả một PDA

Để chấp nhận ngôn ngữ L như trên ta dùng bộ điều khiển có hai trạng thái q1, q2 và một Stack trên đó ta đặt các đĩa xanh (B), vàng (Y), đỏ (R). Thiết bị sẽ thao tác theo các quy tắc sau đây:

1) Máy sẽ bắt đầu với một đĩa đỏ ở trên Stack và bộ điều khiển ở trạng thái q1.

2) Nếu 0 được đưa vào thiết bị thì ta đặt một đĩa xanh vào Stack. Nếu đưa 1 vào thiết bị ở trạng thái q1 thì ta đặt một đĩa vàng vào Stack. Cả hai trường hợp thiết bị không thay đổi trạng thái.

3) Nếu c được đưa vào thiết bị ở trạng thái q1 thì thiết bị đổi trạng thái sang q2 và không thay đổi Stack.

4) Nếu 0 được đưa vào thiết bị ở trạng thái q2 và đỉnh Stack là đĩa màu xanh thì đĩa được lấy ra. Nếu 1 đưa vào thiết bị ở trạng thái q2 và đĩa vàng tại đỉnh Stack thì ta loại bỏ đĩa này. Trạng thái q2 không thay đổi.

5) Nếu thiết bị ở trạng thái q2 và đĩa đỏ tại đỉnh Stack ta loại bỏ đĩa này không cần ký hiệu nhập.

6) Ngoài các trường hợp trên thì thiết bị không thay đổi.

Các quy tắc trên được tóm tắt như trong bảng sau:

INPUT				
Đỉnh Stack	Trạng thái	0	1	c
Xanh	q1	Thêm đĩa xanh, giữ nguyên q1	Thêm đĩa vàng, giữ nguyên q1	Chuyển sang q2
	q2	Xoá đỉnh Stack, giữ		

		nguyên q2		
Vàng	q1	Thêm đĩa xanh, giữ nguyên q1	Thêm đĩa vàng, giữ nguyên q1	Chuyển sang q2
	q2		Xoá đỉnh Stack giữ nguyên q2	
Đỏ	q1	Thêm đĩa xanh, giữ nguyên q1	Thêm đĩa vàng, giữ nguyên q1	Chuyển sang q2
	q2	Xoá đỉnh Stack không cần đọc input		

Hình 6.2 - Mô tả hoạt động của PDA chấp nhận ngôn ngữ {wcwR |w (0+1)*}

Một chuỗi được chấp nhận bởi thiết bị nếu nó đã đọc duyệt qua đến hết chuỗi đồng thời với Stack trở về trạng thái rỗng.

Nhận xét : Nhờ Stack có khả năng lưu giữ một số bất kỳ các ký hiệu mà PDA có thể nhớ nửa phần đầu của chuỗi (w) cho tới khi gặp ký hiệu phân cách c, cho dù chuỗi có độ dài lớn đến bao nhiêu. Và sau đó, các ký hiệu này được đem ra để so sánh dần với phần chuỗi ngược còn lại (wR). Một ôtômát hữu hạn không có được khả năng ghi nhớ đó.

Định nghĩa

Ôtômát đẩy xuống có một bộ điều khiển hữu hạn và một Stack. Stack chứa một chuỗi các ký hiệu thuộc một bộ chữ cái nào đó. Ký hiệu bên trái nhất của chuỗi xem như ký hiệu tại đỉnh Stack. PDA không đơn định nếu như có một số hữu hạn các lựa chọn phép chuyển trong mỗi lần chuyển.

Phép chuyển sẽ có hai kiểu:

- Kiểu thứ nhất phụ thuộc vào ký hiệu nhập, tức là với một trạng thái, một ký hiệu tại đỉnh Stack và một ký hiệu nhập; PDA sẽ lựa chọn trạng thái kế tiếp và một chuỗi các ký hiệu thay thế trên Stack, đầu đọc dịch đi sang phải một ký hiệu.

- Kiểu thứ hai không phụ thuộc vào ký hiệu nhập, gọi là - dịch chuyển : tương tự như kiểu thứ nhất, chỉ ngoại trừ là ký hiệu nhập không được dùng và đầu đọc không dịch chuyển sau khi chuyển trạng thái. Thực chất, bước chuyển đặc biệt này là một sự tạm ngừng quan sát trên băng nhập để sắp xếp lại các ký hiệu trong ngăn xếp.

Có hai cách để định nghĩa ngôn ngữ chấp nhận bởi ôtômát đẩy xuống:

- Ngôn ngữ được chấp nhận bởi Stack rỗng: gồm tất cả các input mà sau một chuỗi các phép chuyển trạng thái làm cho ôtômát dẫn tới Stack rỗng.

- Ngôn ngữ được chấp nhận bởi trạng thái kết thúc: ta thiết kế một số trạng thái kết thúc, khi đó ngôn ngữ chấp nhận bởi ôtômát có thể định nghĩa gồm tất cả các input mà có một chuỗi các phép chuyển làm cho ôtômát dẫn tới một trong những trạng thái kết thúc.

Ta có thể thấy hai cách định nghĩa cho sự chấp nhận chuỗi này là tương đương nhau trong mọi trường hợp, có nghĩa là nếu một tập hợp được chấp nhận bởi Stack rỗng của một PDA nào đó thì cũng sẽ được chấp nhận bằng trạng thái kết thúc trên một PDA khác, và ngược lại. Thiết kế PDA chấp nhận chuỗi bằng trạng thái kết thúc thường phổ biến hơn, nhưng sẽ dễ dàng hơn để chứng minh nguyên lý cơ bản của PDA khi thiết kế PDA chấp nhận chuỗi bằng Stack rỗng. Nguyên lý này được phát biểu như sau: Một ngôn ngữ được chấp nhận bởi PDA khi và chỉ khi nó là một ngôn ngữ phi ngữ cảnh.

Một cách hình thức, ta định nghĩa:

Định nghĩa : Một ôtômát đẩy xuống M là một hệ thống M (Q, , , , q0, Z0, F), trong đó :

1) Q là tập hữu hạn các trạng thái

2)   là bộ chữ cái gọi là bộ chữ cái nhập

3)   là bộ chữ cái gọi là bộ chữ cái Stack

4) : hàm chuyển ánh xạ từ Q ({ })   tập con hữu hạn của Q   *

5) q0 là trạng thái khởi đầu

5) Z0 là một chữ cái riêng của Stack gọi là ký hiệu bắt đầu trên Stack

6) F   Q là tập các trạng thái kết thúc

(Trong trường hợp PDA được thiết kế chấp nhận ngôn ngữ bằng Stack rỗng thì tập hợp F =)

Trừ khi ta dùng các ký hiệu khác, ta quy ước dùng chữ nhỏ gần đầu bảng chữ cái để chỉ các ký hiệu nhập, các chữ nhỏ cuối bảng chữ cái để chỉ các chuỗi nhập. Các chữ hoa và chữ Hy lạp chỉ ký hiệu và chuỗi ký hiệu Stack.

Câu hỏi :

?

So sánh các thành phần trong định nghĩa hình thức cho một ôtômát đẩy xuống

PDA với ôtômát hữu hạn đã khảo sát trong chương 3 ? Nêu những khả năng mở

rộng của PDA so với FA ?

Sự dịch chuyển

Hàm chuyển phụ thuộc ký hiệu nhập :

(q, a, Z) = {(p1, 1), (p2, 2), ..., (pm, m)}

trong đó q và pi, 1 i m, là các trạng thái thuộc tập Q, a , Z là một ký hiệu Stack và i *, 1 i m, là PDA ở trạng thái q với ký hiệu nhập a và ký hiệu Z tại đỉnh Stack, nó đi vào một trạng thái pi nào đó thay Z bằng i và dịch chuyển đầu đọc đi một ký hiệu. Ta quy ước rằng ký hiệu bên trái nhất của i sẽ là ký hiệu được thay cao nhất trên Stack (nghĩa là nó nằm tại đỉnh Stack mới) và ký hiệu bên phải nhất của i là ký hiệu được thay thấp nhất trong Stack. Chú ý rằng không được phép chọn pi và j với i j tại một bước chuyển nào đó.

Hàm chuyển không phụ thuộc ký hiệu nhập :

(q, , Z) = {(p1, 1), (p2, 2), ..., (pm, m)}

trong đó q là trạng thái mà PDA đang giữ, độc lập với ký hiệu nhập, PDA đi vào trạng thái pi thay Z bởi i với 1 i m. Trong trường hợp này đầu đọc không dịch chuyển.

Thí dụ 6.1 :Mô tả dưới đây cho PDA chấp nhận ngôn ngữ {wcwR w (0+1)*} bằng Stack rỗng.

M ({q1, q2}, {0, 1, c}, {R, B, Y}, d, q1, R, Æ)

1) d(q1, 0, R) = {(q1, BR)}

2)d(q1, 1, R) = {(q1, YR)}

3)d(q1, 0, B) = {(q1, BB)}

4)d(q1, 1, B) = {(q1, YB)}

5)d(q1, 0, Y) = {(q1, BY)}

6)d(q1, 1, Y) = {(q1, YY)}

7)d(q1, c, R) = {(q2, R)}

8)d(q1, c, B) = {(q2, B)}

9)d(q1, c, Y) = {(q2, Y)}

10)d(q2, 0, B) = {(q2, e)}

11)d(q2, 1, Y) = {(q2, e)}

12)d(q2, e, R) = {(q2, e)}

Hình 6.3 - Mô tả PDA chấp nhận wcwR bằng Stack rỗng

Chú ý rằng mỗi phép chuyển PDA sẽ viết lên đỉnh Stack một chuỗi có độ dài 2. Chẳng hạn d(q1, 0, R) = {(q1, BR)}. Nếu có độ dài bằng 1 thì PDA đơn giản là thay ký hiệu tại đỉnh Stack và không làm thay đổi độ dài Stack. Nếu bằng thì PDA loại bỏ (Pop) phần tử tại đỉnh Stack. Chẳng hạn d(q2, e, R) = {(q2, e)} nghĩa là PDA ở trạng thái q2 với R ở đỉnh Stack thì PDA xóa R độc lập với ký hiệu nhập, trong trường hợp này đầu đọc không dịch chuyển, điều này có nghĩa là PDA không yêu cầu còn một ký hiệu nào trên chuỗi nhập.

Hình thái (ID : Instantaneous Descriptions)

Để hình thức hóa cấu hình của một PDA với một PDA cụ thể, ta định nghĩa một hình thái (ID). ID phải ghi nhớ trạng thái và nội dung của Stack. ID là một bộ ba (q, w,), trong đó q là trạng thái, w là chuỗi nhập và là chuỗi các ký hiệu Stack.

Nếu M (Q, , , , q0, Z0, F) là một PDA, ta nói :

(q, aw, Z) ⊢M (p, w,) nếu (q, a, Z) chứa (p,)

Lưu ý a có thể là một ký hiệu trong input hoặc . Chẳng hạn với PDA mô tả như trên, ta có (q1, BY) d(q1, 0, Y), suy ra rằng (q1, 011, YYR) ⊢ (q1, 11, BYYR).

Ta dùng ký hiệu ⊢*M cho bao đóng phản xạ và bắc cầu của ⊢M, tức là : I ⊢* I đối với mỗi ID I, và I ⊢M J và J ⊢*M K thì I ⊢*M K. Ta viết I ⊢i

K nếu ID I trở thành K sau chính xác i bước chuyển. Chữ chỉ số dưới M trong các ky 1hiệu ⊢M, ⊢iM và ⊢*M có thể được bỏ qua khi M đã được xác định.

Ngôn ngữ chấp nhận bởi PDA

Với PDA M (Q, , , , q0, Z0, F), ta định nghĩa :

Ngôn ngữ được chấp nhận bởi trạng thái kết thúc là:

L (M) = {w (q0, w, Z0) ⊢* (p, ,) với p F và *}

Ngôn ngữ được chấp nhận bởi Stack rỗng là :

N(M) = {w (q0, w, Z0) ⊢* (p, ,) với p Q}.

Khi có sự chấp nhận bằng Stack rỗng thì tập trạng thái kết thúc là không còn cần thiết vì vậy ta ký hiệu tập trạng thái kết thúc F là .

Thí dụ 6.2 :Các phép chuyển hình thái của PDA chấp nhận chuỗi 001c100 thuộc ngôn ngữ {wcwR w (0+1)*} bằng Stack rỗng như sau :

(q1, 001c100, R) ⊢ (q1, 01c110, YR) ⊢ (q1, 1c110, YYR) ⊢ (q1, c100, BYYR) ⊢

(q2, 100, BYYR) ⊢ (q2, 00, YYR) ⊢ (q2, 0, YR) ⊢ (q2, , R) ⊢ (q2, ,) : Chấp nhận

Nhận xét : Trong ví dụ thiết kế PDA chấp nhận ngôn ngữ {wcwR w (0+1)*} bằng Stack rỗng như trên, ta thấy các giá trị hàm chuyển của nó luôn là là đơn trị. Tại mỗi thời điểm từ một trạng thái trong bộ điều khiển, có thể đọc vào hoặc không đọc một ký hiệu trên băng nhập, với một ký hiệu tại đỉnh Stack, chỉ có một giá trị xác định bước chuyển kế tiếp. Vì thế, ta gọi dạng PDA này là ôtômát đẩy xuống đơn định - DPDA.

PDA không đơn định (NPDA)

Thí dụ 6.3 : Thiết kế PDA chấp nhận ngôn ngữ {wwR | w   (0+1)*} bằng Stack rỗng.

Câu hỏi :

?

Hãy nêu vai trò của ký hiệu c trong ngôn ngữ được cho bởi thí dụ 6.1 và cho nhận

xét về sự khác biệt dạng chuỗi thuộc ngôn ngữ trong thí dụ 6.3 với thí dụ 6.1 đã nêu ở trên ?

Rõ ràng ta thấy khi không có sự hiện diện của ký hiệu c ở giữa chuỗi nhập để xác định thời điểm bộ điều khiển có thể chuyển từ trạng thái q1 sang trạng thái q2, thì vấn đề sẽ trở nên phức tạp hơn khi cần phải quyết định đâu là ký hiệu bắt đầu cho chuỗi ngược (wR) ? Ở mỗi thời điểm mà bộ điều khiển đọc thấy hai ký hiệu liên tiếp giống nhau trong chuỗi nhập thì bắt buộc nó phải đoán thử cả hai khả năng cho ký hiệu thứ hai: hoặc vẫn giữ trạng thái q1 và Push vào Stack nếu xem ký hiệu này vẫn thuộc chuỗi xuôi (w), hoặc chuyển sang trạng thái q2 và Pop khỏi Stack nếu xem nó là ký hiệu bắt đầu cho chuỗi ngược (wR).

Mô tả PDA không đơn định chấp nhận ngôn ngữ {wwR w   (0+1)*} bằng Stack rỗng

M ({q1, q2}, {0, 1}, {R, B, Y}, d, q1, R, Æ)

1)d(q1, 0, R) = {(q1, BR)}

2)d(q1, 1, R) = {(q1,YR)}

3)d(q1, 0, B) = {(q1, BB), (q2, e)}

4)d(q1, 0, Y) = {(q1, BY)}

5)d(q1, 1, B) = {(q1, YB)}

6)d(q1, 1, Y) = {(q1, YY),(q2, e)}

7)d(q2, 0, B) = {(q2, e)}

8)d(q2, 1, Y) = {(q2, e)}

9)d(q1, e, R) = {(q2, e)}

10)d(q2, e, R) = {(q2, e)}

Hình 6.4 - Mô tả PDA không đơn định chấp nhận wwR bằng Stack rỗng

Quy tắc (1) đến (3) cho phép M lưu trữ input trên Stack, quy tắc (3) và (6) cho phép M lựa chọn một trong hai phép chuyển. M có thể quyết định (đoán) đã đi đến giữa chuỗi nó chuyển sang phép chuyển thứ 2: M chuyển sang q2 và thử sự thích hợp của phần chuỗi còn lại với các ký hiệu đang ở trên Stack. Nếu M đoán đúng và nếu chuỗi nhập có dạng wwR thì M sẽ làm rỗng Stack của nó và chấp nhận chuỗi nhập.

Cũng như NFA một PDA không đơn định (NPDA) M chấp nhận một input nếu có một chuỗi các lựa chọn mà M làm rỗng Stack của nó. Nghĩa là M luôn luôn "đoán đúng", đoán sai không phải là nguyên nhân để loại bỏ input. Một input bị loại bỏ nếu và chỉ nếu không có sự lựa chọn nào để làm rỗng Stack (hay là không thể "đoán đúng" vì không tồn tại cách đúng).

Thí dụ 6.4 :Các phép chuyển hình thái của PDA chấp nhận chuỗi 001100 thuộc ngôn ngữ {wwR w   (0+1)*} bằng Stack rỗng như sau :

Khởi đầu

_

(q1, 001100, R) ® (q2, 001100, e) : Không chấp nhận

_

(q1, 01100, BR) ® (q2, 1100, R) ® (q2, 1100, e) : Không chấp nhận

–

(q1, 1100, BBR)

–

(q1, 100, YBBR) ® (q2, 00, BBR)

– –

(q1, 00, YYBBR) (q2, 0, BR) (q2, e, R) (q2, e, e) : Chấp nhận

–

(q1, 0, BYYBBR) ® (q2, e, YYBBR) : Không chấp nhận

–

(q1, e, BBYYBBR) : Không chấp nhận

Hình 6.5 - Hình thái của PDA với input 001100

PDA đơn định (DPDA)

Một PDA M (Q, , , , q0, Z0, F) được gọi là đơn định nếu:

1) q Q và Z : nếu (q, , Z) thì (q, a, Z) = , a

2) Không có q Q, Z và a ({ }) mà (q, a, Z) chứa nhiều hơn một phần tử.

Điều kiện 1 không cho phép khả năng chọn lựa giữa phép chuyển không xác định ký hiệu nhập (- dịch chuyển) và phép chuyển trên một ký hiệu input. Điều kiện 2 không cho phép chọn lựa một vài phép chuyển nào đó (q, a, Z) hay (q, , Z). Không như ôtômát hữu hạn FA, một PDA thì thông thường được xét là không đơn định trừ khi ta có ghi chú cụ thể.

Đối với ôtômát hữu hạn, dạng đơn định và không đơn định là tương đương nhau về phương diện chấp nhận ngôn ngữ. Tuy nhiên, điều này không đúng với ôtômát đẩy xuống, PDA không đơn định và PDA đơn định là không tương đương nhau. Thực tế ngôn ngữ wwR được chấp nhận bởi một PDA không đơn định nhưng không được chấp nhận bởi bất kỳ một PDA đơn định nào.

PDA VÀ VĂN PHẠM PHI NGỮ CẢNH

Tương đương của việc chấp nhận chuỗi bởi trạng thái kết thúc và bởi Stack rỗng

ĐỊNH LÝ 6.1: Nếu L là L(M2) với PDA M2 thì L là N(M1) với PDA M1 nào đó.

Chứng minh

Ta sẽ xây dựng M1 tương tự như M2 nhưng M1 sẽ xóa rỗng Stack của nó khi M2 đi vào trạng thái kết thúc. Ta dùng một trạng thái qe của M1 để xóa Stack của nó và dùng ký hiệu đánh dấu đáy Stack M1 bằng ký hiệu X0, vì vậy M1 không thể làm rỗng Stack của nó khi M2 chưa đi vào trạng thái kết thúc.

Đặt M2 (Q, , , , q0, Z0, F) là PDA sao cho L = L(M2).

Đặt M1 (Q {qe, q0’}, , , ’, q0’, X0,) trong đó ’ định nghĩa như sau:

1) ’(q0’, e, X0) = {(q0, Z0X0)}

2) ’(q, a, Z) chứa mọi phần tử của (q, a, Z), q Q, a hoặc a = và Z

3) q F và Z {X0}, ’(q, , Z) chứa (qe,)

4) Z {X0}, ’(q0’, e, Z) chứa (qe,)

Quy tắc 1 làm cho PDA M1 đi vào trạng thái khởi đầu của M2 trừ việc thêm X0 vào đáy Stack. Quy tắc 2 cho phép M1 chuyển tương tự như M2. Quy tắc 3 và 4 cho phép M1 chọn việc đi vào trạng thái qe và xoá Stack hay là tiếp tục mô phỏng M2. Chú ý rằng M2 có thể xóa rỗng Stack của nó khi chưa tới trạng thái kết thúc vì vậy M1 phải được đánh dấu đáy Stack bằng X0. Vì nếu không làm như vậy thì khi M1 chuyển tương tự như M2, M1 sẽ xoá rỗng Stack và chấp nhận input trong khi M2 chưa đi vào trạng thái kết thúc nghĩa là input chưa được chấp nhận.

Đặt x   L(M2) thì (q0, x, Z0) ⊢*M2 (q, ,) với q   F. Ta xét M1 với input x.

Theo quy tắc 1 : (q0’, x, X0) ⊢*M1 (q0, x, Z0X0)

Theo quy tắc 2 mỗi phép chuyển của M2 là một phép chuyển trong M1, vậy:

(q0, x, Z0) ⊢*M1 (q, e, g)

Nếu một PDA có thể thực hiện một chuỗi các phép chuyển từ một ID đã cho thì nó có thể làm một chuỗi các phép chuyển đó từ một ID bất kỳ thu được từ ID đầu tiên bằng cách thêm các chuỗi ký hiệu Stack vào dưới chuỗi Stack ban đầu (vì các ký hiệu ở phía dưới của Stack không làm ảnh hưởng gì).

Vậy (q0’, x, X0) ⊢M1 (q0, x, Z0X0) ⊢*M1 (q, , X0).

Theo quy tắc 3 và 4 : (q, , X0) ⊢*M1 (qe, ,).

Vì vậy (q0’, x, X0) ⊢*M1 (qe, ,) và M1 chấp nhận chuỗi x bằng Stack rỗng.

Ngược lại, nếu M1 chấp nhận x bằng Stack rỗng thì dễ dàng chỉ ra rằng chuỗi các phép chuyển phải bắt đầu bằng một phép chuyển theo quy tắc 1, sau đó bằng một chuỗi phép chuyển theo quy tắc 2, trong khi thực hiện các phép chuyển này M1 chuyển tương tự như M2, sau đó xóa Stack của M1 bằng quy tắc chuyển 3 và 4.

Vậy x   L(M2).

ĐỊNH LÝ 6.2 : Nếu L là N(M1) với PDA M1 nào đó thì L là L(M2) với một PDA M2 nào đó.

Chứng minh

Ta sẽ xây dựng M2 tương tự M1 và M2 đi vào trạng thái kết thúc khi và chỉ khi M1 làm rỗng Stack của nó.

Đặt M1 (Q, , , , q0, Z0, F) là PDA sao cho L = N(M1).

Đặt M2 (Q   {q0', qf}, ,   {X0}, ', q0', X0, {qf}) trong đó ' được định nghĩa như sau:

1. '(q0', e, X0) = {(q0, Z0X0)}
2. q   Q, a   { }, và Z   : '(q, a, Z) = (q, a, Z)

3) q   Q, '(q, , X0) chứa (qf,)

Quy tắc 1 cho phép M2 đi vào hình thái khởi đầu ID của M1, trừ việc M2 sẽ có chứa ở dưới đáy Stack của nó ký hiệu X0, ký hiệu này sẽ nằm bên dưới tất cả các ký hiệu Stack của M1. Quy tắc 2 cho phép M2 chuyển tương tự như M1. Khi M1 làm rỗng Stack của nó, thì M2 khi chuyển tương tự như M1 sẽ xóa toàn bộ Stack của nó trừ ký hiệu X0 nằm dưới đáy Stack. Quy tắc 3 làm cho M2 sau đó khi gặp X0 xuất hiện thì đi vào trạng thái kết thúc và chấp nhận input x.

Chứng minh L(M2) = N(M1) cũng tương tự như định lý 6.1

Tương đương giữa PDA và CFL

ĐỊNH LÝ 6.3: Nếu L là ngôn ngữ phi ngữ cảnh thì tồn tại PDA M sao cho L = N(M).

Chứng minh

Giả sử không thuộc L(G) (có thể sửa đổi lý luận cho trường hợp ngôn ngữ L(G) có chứa). Đặt G (V, T, P, S) là văn phạm phi ngữ cảnh có dạng chuẩn Greibach sinh ra L. Đặt M ({q}, T, V, , q, S,), trong đó (q, a, A) chứa (q,) khi và chỉ khi A a là một luật sinh trong P.

PDA M mô phỏng chuỗi dẫn xuất trái của G. Vì G là dạng chuẩn Greibach nên mỗi dạng câu trong dẫn xuất trái gồm một chuỗi các ký hiệu kết thúc x sau đó là một chuỗi các biến . M lưu trữ phần hậu tố của dạng câu bên trái trên Stack của nó sau khi xử lý phần tiền tố x.

Một cách hình thức ta chỉ ra rằng :

S * x bằng dẫn xuất trái khi và chỉ khi (q, x, S) ⊢*M (q, ,) (1)

Trước tiên, chúng ta giả sử (q, x, S) ⊢i (q, ,) và sẽ chỉ ra bằng quy nạp theo số lần i rằng S * x .

Với i = 0, điều đó hiển nhiên đúng vì x = và = S.

Giả sử i 1 và đặt x = ya.

Xét bước chuyển hình thái trước bước cuối :

(q, ya, S) ⊢i -1 (q, a,) ⊢ (q, ,)(2)

Nếu loại bỏ ký hiệu a ở cuối chuỗi input trong hình thái đầu tiên của (2), ta có: (q, y, S) ⊢i -1 (q, ,) (vì a không ảnh hưởng đến các phép chuyển của M).

Theo giả thiết quy nạp S * y . Phép chuyển (q, a,) ⊢ (q, ,) sẽ suy ra = A , với A V và A a là một luật sinh trong G và = .

Vậy S * y ya = x

Ta đã chứng minh xong "nếu" của giả thiết (1)

Ngược lại, ta giả sử S i x bằng dẫn xuất trái. Ta sẽ chứng minh quy nạp theo số bước dẫn xuất i rằng: (q, x, S) ⊢* (q, ,)

Với i = 0: phép chuyển hiển nhiên đúng

Xét i 1 và giả sử S i -1 yA   ya , trong đó x = ya và = . Theo giả thiết quy nạp : (q, y, S) ⊢* (q, , A). Vậy (q, ya, S) ⊢* (q, a, A)

Vì A a là một luật sinh nên (q, a, A) chứa (q,). Vậy :

(q, x, S) ⊢* (q, a, A) ⊢* (q, ,)

Hay phần "chỉ nếu" của giả thiết (1) cũng đã được chứng minh xong.

Để kết thúc việc chứng minh, ta chú ý rằng giả thiết (1) với = thì S * x nếu và chỉ nếu (q, x, S) ⊢* (q, ,). Tức là x L(G) khi và chỉ khi x N(M).

Thí dụ 6.3 :Xây dựng NPDA chấp nhận ngôn ngữ sinh bởi CFG G có các luật sinh như sau :

S ® aAA

A ® aS | bS | a

Ta có : CFG G ({S, A}, {a, b}, P, S)

NPDA tương đương M ({q}, {a, b}, {S, A}, , q, S,) với như sau :

1) d (q, a, S) = {(q, AA)}

2) d (q, a, A) = {(q, S), (q, e)}

3)d (q, b, A) = {(q, e)}

ĐỊNH LÝ 6.4 : Nếu L là N(M) với PDA M thì L và ngôn ngữ phi ngữ cảnh.

Chứng minh

Gọi PDA M (Q, , , , q0, Z0,). Đặt G (V, , P, S) là CFG, trong đó :

. V là tập các đối tượng dạng [q, A, p] với p, q Q; A

. S là ký hiệu chưa kết thúc mới thêm vào.

. P là tập các luật sinh có dạng :

1) S ® [q0, Z0, q] , q Î Q.

2) [q, A, q m+1] ® a[q1, B1, q2][q2, B2, q3] ... [qm, Bm, qm+1]

 q, q1, q2, ..., qm+1 Q, a   { } và A, B1, B2, ..., Bm

sao cho (q, a, A) có chứa (q1, B1B2 .. Bm).

Nếu m = 0 thì luật sinh có dạng [q, A, q1] a.

Để nắm được chứng minh, cần phải lưu ý rằng các biến và luật sinh trong G được xác định sao cho dẫn xuất trái trong G của x mô phỏng PDA khi cho x nhập vào. Cụ thể hơn, các biến xuất hiện tại một bước bất kỳ trong G tương đương với các ký hiệu trên Stack của M. Nói cách khác [q, A, p] dẫn ra x nếu và chỉ nếu x là nguyên nhân làm M xoá rỗng Stack của nó bằng chuỗi các phép chuyển từ trạng thái q đến trạng thái p.

Để chứng minh L(G) = N(M), ta quy nạp theo số bước dẫn xuất của G hoặc số bước chuyển trạng thái của M rằng [q, A, p] *G x nếu và chỉ nếu (q, x, A) ⊢*M (p, ,)

Thí dụ 6.4 :Xây dựng CFG G tương đương sinh ra ngôn ngữ được chấp nhận bởi PDA sau :

M ({q0, q1}, {0, 1}, {Z0, X}, d, q0, Z0, Æ)

với d được cho như sau :

1) d (q0, 0, Z0) = {(q0, XZ0)}

2) d (q0, 0, X) = {(q0, XX)}

3) d (q0, 1, X) = {(q1,)}

4) d (q1, 1, X) = {(q1,)}

5) d (q1, , x) = {(q1,)}

6) d (q1, , Z0) = {(q1,)}

Ta xây dựng CFG G (V, {0, 1}, P, S) sinh ra N(M) với các thành phần như sau :

V = { S, [q0, X, q0], [q0, X, q1], [q1, X, q0], [q1, X, q1],

[q0, Z0, q0], [q0, Z0, q1], [q1, Z0, q0], [q1, Z0, q1] }

Tập luật sinh P chứa các luật sinh có dạng :

Các luật sinh cho ký hiệu bắt đầu S : S ® [q0, Z0, q0] | [q0, Z0, q1]

Các luật sinh cho các biến khác trong V được xây dựng từ các hàm chuyển của PDA như sau :

d1) [q0, Z0, q0] ® 0 [q0, X, q0][q0, Z0, q0]

| 0 [q0, X, q1][q1, Z0, q0]

[q0, Z0, q1] ® 0 [q0, X, q0][q0, Z0, q1]

| 0 [q0, X, q1][q1, Z0, q1]

d2) [q0, X, q0] ® 0 [q0, X, q0][q0, X, q0]

| 0 [q0, X, q1][q1, X, q0]

[q0, X, q1] ® 0 [q0, X, q0][q0, X, q1]

| 0 [q0, X, q1][q1, X, q1]

d3) [q0, X, q1] ® 1

d4) [q1, Z0, q1] ®

d5) [q1, X, q1] ®

d6) [q1, X, q1] ® 1

Nhận xét rằng không có luật sinh nào cho các biến [q1, X, q0] và [q1, Z0, q0]. Vì tất cả các luật sinh cho biến [q0, X, q0] và [q0, Z0, q0] đều có chứa [q1, X, q0] hoặc [q0, Z0, q0] ở vế phải, nên sẽ không thể có chuỗi ký hiệu kết thúc nào có thể được dẫn ra từ các biến [q0, X, q0] hoặc [q0, Z0, q0]. Loại bỏ 4 biến này ra khỏi tập biến V và xóa các luật sinh có liên quan đến chúng trong tập P, ta thu được văn phạm có dạng như sau:

S ® [q0, Z0, q1]

[q0, Z0, q1] ® 0 [q0, X, q1][q1, Z0, q1]

[q0, X, q1] ® 0 [q0, X, q1][q1, X, q1]

[q0, X, q1] ® 1

[q1, Z0, q1] ®

[q1, X, q1] ®

[q1, X, q1] ® 1

Câu hỏi :

?

Sinh viên hãy dùng các kiến thức đã học trong chương trước (ĐỊNH LÝ 5.2)

để viết một văn phạm tương đương với văn phạm trên không có chứa các ký

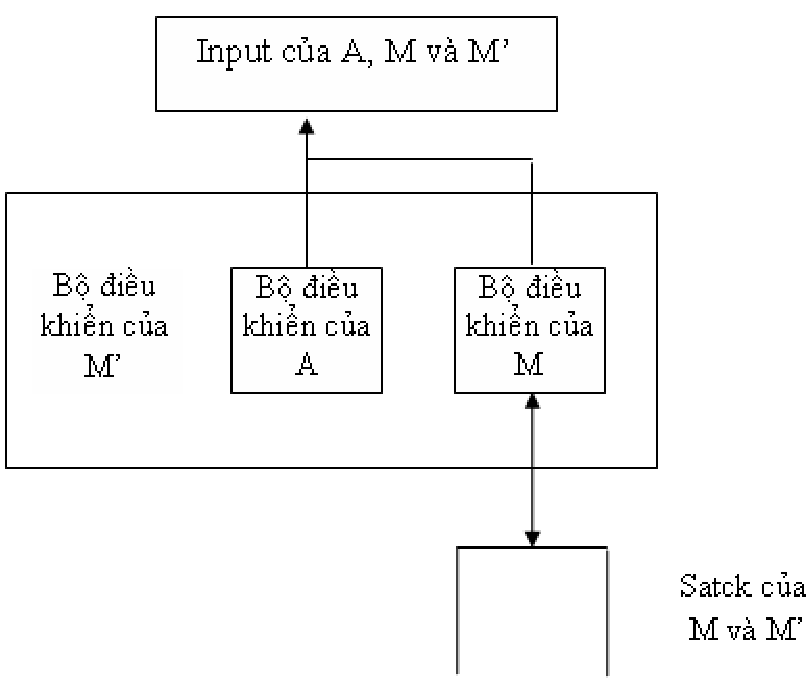
hiệu vô ích ?

Quan hệ giữa CFL và tập hợp chính quy

ĐỊNH LÝ 6.5 :Nếu L là CFL và R là tập chính quy thì L R là CFL.

Chứng minh

Đặt L là L(M) với PDA M (QM, , , M, q0, Z0, FM) và đặt R là L(A) với DFA A (QA, å, dA, p0, FA). Ta xây dựng PDA M' cho L R bằng cách cho M và A cùng "chạy song song". Tức là với một ký hiệu nhập a thì M và A thực hiện các phép chuyển độc lập nhau. M' chấp nhận input nếu cả M và A cùng chấp nhận.



Hình 6.6 - Chạy một FA và PDA song song

Một cách hình thức, đặt M’ (QA   QM, , , , [p0, q0], Z0, FA   FM), trong đó hàm chuyển   được xác định như sau :

  ([p, q], a, X) chứa ([p’, q’],)   dA(p, a) = p’ và dM(q, a, X) chứa (q’,).

Chú ý rằng a có thể bằng , khi đó p’ = p.

Dễ dàng chứng minh quy nạp theo i rằng : ([p0, q0], w, Z0) ⊢iM ’ ([p, q], ,)   (q0, w, Z0) ⊢iM (q, ,) và d(p0, w) = p (1)

Với i = 0: thì (1) hiển nhiên đúng vì p = p0, q = q0,   = Z0 và w = .

Giả sử (1) đúng tới i - 1 (i > 0).

Xét ([p0, q0], xa, Z0) ⊢i -1M ’ ([p’, q’], a,) ⊢M ’ ([p, q], ,) , trong đó w = xa và a là   hoặc là một ký hiệu   .

Theo giả thiết quy nạp, dA(p0, x) = p’ và (q0, x, Z0) ⊢i -1M (q’, ,).

Theo định nghĩa của d, thực tế ([p’, q’], a,) ⊢M ‘ ([p, q], ,) nên có thể suy ra dA(p’, a) = p và (q’, a,) ⊢M (q, ,). Vậy dA(p0, w) = p và (q0, w, Z0) ⊢*M (q, ,).

Tương tự, ta có thể chứng minh rằng : Nếu (q0, w, Z0) ⊢iM (q, ,) và dA(p0, w) = p thì ([p0, q0], w, Z0) ⊢*M ’ ([p, q], ,) (xem phần này như bài tập).

Tổng kết chương VI: Đến chương này, chúng ta đã có thể nắm bắt được một vài ý tưởng cơ bản liên quan đến các khái niệm về ngôn ngữ chính quy, ngôn ngữ phi ngữ cảnh, và mối quan hệ của chúng với các dạng ôtômát hữu hạn và đẩy xuống. Những khảo sát chứng tỏ ngôn ngữ chính quy thực sự là một tập hợp con của ngôn ngữ phi ngữ cảnh, và do đó, ôtômát đẩy xuống xét về một mặt nào đó có khả năng nhận dạng ngôn ngữ mạnh hơn rất nhiều so với ôtômát hữu hạn. Điều này gợi cho chúng ta một ý tưởng có thể mở rộng hơn nữa về khả năng đoán nhận ngôn ngữ của cơ chế ôtômát. Nếu so sánh ôtômát hữu hạn và ôtômát đẩy xuống, ta thấy rằng bản chất của sự khác biệt thể hiện ở bộ lưu trữ tạm

thời dùng Stack. Nếu không có bộ lưu trữ, chúng ta có dạng ôtômát hữu hạn, nếu bộ lưu trữ là Stack, ta có dạng ôtômát đẩy xuống mạnh hơn. Từ suy luận này, chúng ta hoàn toàn có thể mong đợi để định nghĩa ngay cả những họ ngôn ngữ rộng lớn hơn nếu có thể cung cấp cho cơ chế ôtômát một bộ nhớ với khả năng lưu trữ linh hoạt hơn. Điều này dẫn đến khái niệm cơ bản về máy Turing sẽ được giới thiệu trong chương sau, một cơ chế ôtômát có tính máy móc hay tính giải thuật.

BÀI TẬP CHƯƠNG VI

6.1. Xây dựng PDA chấp nhận các ngôn ngữ :

a) {0m 1m 2n   m, n   1}

b) {ak bl cn dm   m = k + l + n}

c) {w   w   {a, b}* và #a(w) = #b(w)}

d) {w   w   {a,b}* và #a(w) = 2#b(w)}

6.2. Xây dựng PDA tương đương với văn phạm :

a) S ® + SS | *SS | a

b)S ® aS | bS | aA

A ® bB| b

B ® aC

C ® b

6.3. Xây dựng văn phạm CFG tương đương với các PDA sau :

a) M ({q0, q1}, {0, 1}, {Z0, X}, , q0, Z0,), trong đó   được cho như sau:

d(q0, 1, Z0) = {(q0, XZ0)}

d(q0, 0, X) = {(q0, XX)}

d(q0, 1, X) = {(q1, e)}

d(q1, 1, X) = {(q1, e)}

d(q1, e, X) = {(q1, e)}

d(q1, e, Z0) = {(q1, e)}

b) M ({q0, q1}, {0, 1}, {Z0, X}, , q0, Z0,), trong đó được cho như sau:

d(q0, 1, Z0) = {(q0, XZ0)}

d(q0, 1, X) = {(q0, XX)}

d(q0, 0, X) = {(q1, X)}

d(q0, e, Z0) = {(q0, e)}

d(q1, 1, X) = {(q1, e)}

d(q1, 0, Z0) = {(q0, Z0)}

c) M ({q0, q1}, {a, b, c}, {Z0, X}, , q0, Z0,), trong đó được cho như sau:

d(q0, a, Z0) = {(q0, X)}

d(q0, a, X) = {(q0, XX)}

d(q0, c, X) = {(q1, X)}

d(q0, b, Z0) = {(q0, X)}

d(q0, b, X) = {(q0, XX)}

d(q1, c, X) = {(q1, e)}

Văn phạm chính quy và các tính chất
Nội dung chính : Trong chương này, ta sẽ đề cập đến lớp văn phạm chính quy (dạng văn phạm tuyến tính trái hoặc phải) - một phương tiện khác để xác định ngôn ngữ và ta lại thấy rằng lớp ngôn ngữ do chúng sinh ra vẫn là lớp ngôn ngữ chính quy. Điều này được thể hiện bởi mối tương quan giữa văn phạm chính quy và ôtômát hữu hạn. Tiếp sau đó, ta sẽ nghiên cứu một số tính chất của lớp ngôn ngữ chính quy, cũng như các giải thuật xác định tập chính quy. Mục tiêu cần đạt: Cuối chương, sinh viên cần phải nắm vững : Định nghĩa một biểu thức chính quy ký hiệu cho tập ngôn ngữ. Mối liên quan giữa ôtômát hữu hạn và biểu thức chính quy. Các tính chất của tập chính quy. Xây dựng ôtômát từ biểu thức chính quy Viết văn phạm chính quy sinh ra cùng tập ngôn ngữ được cho bởi ôtômát. Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, sinh viên cần nắm vững các thành phần tổng quát của một văn phạm cấu trúc, các dạng luật sinh; hiểu biết về ngôn ngữ tự nhiên; cơ chế đoán nhận ngôn ngữ từ ôtômát hữu hạn và cách phát sinh một lớp ngôn ngữ thông qua biểu thức chính quy; … Tài liệu tham khảo : [1] V.J. Rayward-Smith – A First course in Formal Language Theory (Second Editor) – McGraw-Hill Book Company Europe – 1995 (Chapter 3 : Regular Language I) [2] Hồ Văn Quân – Giáo trình lý thuyết ôtômát và ngôn ngữ hình thức – Nhà xuất bản Đại học quốc gia Tp. Hồ Chí Minh – 2002 (Chương 4 : Văn phạm chính quy) [3] From Wikipedia, the free encyclopedia - Regular Grammar: http://en.wikipedia.org/wiki/Regular_grammar

VĂN PHẠM CHÍNH QUY (rg : REGULAR GRAMMAR)

Như trong chương 3 ta đã biết, lớp ngôn ngữ được chấp nhận bởi ôtômát hữu hạn được gọi là ngôn ngữ chính quy và chúng có thể được ký hiệu một cách đơn giản bằng việc dùng một biểu thức chính quy. Chương này giới thiệu một cách khác để mô tả ngôn ngữ chính quy thông qua cơ chế sản sinh ngôn ngữ - đó là văn phạm chính quy.

Xét một định nghĩa cho văn phạm sinh ra các số nguyên không dấu (unsigned interger) bắt đầu bằng một chữ số, theo sau bởi một chuỗi các số (digit sequence) thường dùng trong các ngôn ngữ lập trình như sau:

<digit sequence> ::= 0 1 2 3 4 5 6 7 8 9

0 <digit sequence> 1 <digit sequence>

2 <digit sequence> 3 <digit sequence>

4 <digit sequence> 5 <digit sequence>

6 <digit sequence> 7 <digit sequence>

8 <digit sequence> 9 <digit sequence>

<unsighed integer> ::= 0 1 2 3 4 5 6 7 8 9

1 <digit sequence> 2 <digit sequence>

3 <digit sequence> 4 <digit sequence>

5 <digit sequence> 6 <digit sequence>

7 <digit sequence> 8 <digit sequence>

9 <digit sequence>

Câu hỏi :

?

Bạn có nhận xét gì về dạng chuỗi trong vế phải của các luật sinh văn phạm ?

Trong ví dụ trên, ta thấy mỗi vế phải hoặc là một ký hiệu kết thúc hoặc có dạng của một ký hiệu kết thúc theo sau là một biến. Trong hầu hết mọi ngôn ngữ lập trình, tất cả các ký hiệu cơ bản (số nguyên, tên biến, toán hạng, từ khóa, các ký hiệu hết câu,…) đều có thể định nghĩa bởi những quy luật ngắn dạng này. Vì phần lớn thời gian tiêu tốn trong một trình biên dịch là dùng để nhận dạng các ký hiệu cơ bản, cho nên việc khảo sát lớp văn phạm với các luật sinh dạng như trên là rất cần thiết.

Văn phạm tuyến tính

Một văn phạm G(V, T, P, S) được gọi là tuyến tính trái (left - linear) nếu tất cả các luật sinh của nó có dạng :

A Bw

A w

trong đó A, B là các biến V; w là một chuỗi các ký hiệu kết thúc T* (có thể rỗng).

Một văn phạm G(V, T, P, S) được gọi là tuyến tính phải (right - linear) nếu tất cả các luật sinh của nó có dạng :

A wB

A w

Một văn phạm được gọi là văn phạm chính quy nếu nó thuộc dạng văn phạm tuyến tính trái hoặc tuyến tính phải.

Thí dụ 4.1 : Văn phạm sinh ra các số nguyên không dấu như đã nêu ở trên là văn phạm chính quy vì các luật sinh của nó có dạng tuyến tính phải.

Thí dụ 4.2 : Các văn phạm sau đây là văn phạm chính quy :

Văn phạm G1 ({S}, {a, b}, P1, S) với các luật sinh được cho như sau :

S abS a

là văn phạm tuyến tính phải.

Văn phạm G2 ({S, A, B}, {a, b}, P2, S) với các luật sinh được cho như sau :S Aab

A Aab B

B a

là văn phạm tuyến tính trái.

Thí dụ 4.3 : Ngôn ngữ được ký hiệu bởi biểu thức chính quy 0(10)* được sinh bởi văn phạm tuyến tính phải có các luật sinh sau :

S   0A(1)

A   10A

Và bởi văn phạm tuyến tính trái :

S   S10   0(2)

Sự tương đương giữa văn phạm chính quy và ôtômát hữu hạn

Văn phạm chính quy mô tả tập hợp chính quy trong ngữ cảnh một ngôn ngữ là chính quy khi và chỉ khi nó được sinh ra từ văn phạm tuyến tính trái hoặc văn phạm tuyến tính phải. Kết quả này được xác định bởi hai định lý sau :

ĐỊNH LÝ 4.1 : Nếu L được sinh ra từ một văn phạm chính quy thì L là tập hợp chính quy.

Chứng minh

Trước hết, ta giả sử L = L(G) với một văn phạm tuyến tính phải G(V, T, P, S). Ta xây dựng một NFA có chứa -dịch chuyển M (Q, T, , [S], [ ]) mô phỏng các dẫn xuất trong G.

Q bao gồm các trạng thái có dạng [ ] với   là S hoặc chuỗi hậu tố của vế phải một luật sinh nào đó trong P.

Ta định nghĩa   như sau :

1. Nếu A là một biến, thì ([A],) = {[ ]   A   là một luật sinh}
2. Nếu a thuộc T và   thuộc T*   T*V, thì ([a], a) = {[ ]}

Sau đó, ta có thể dễ dàng chứng minh quy nạp theo độ dài của dẫn xuất rằng ([S], w) chứa [ ] khi và chỉ khi có chuỗi dẫn xuất S * xA xy với A y là một luật sinh trong P và xy = w, hay nếu = S thì w = . Khi [ ] là trạng thái kết thúc duy nhất, M chấp nhận w khi và chỉ khi S * xA w. Nhưng vì mọi chuỗi dẫn xuất cho một chuỗi ký hiệu kết thúc qua ít nhất 1 bước, nên ta thấy rằng M chấp nhận w khi và chỉ khi G sinh ra w. Vì vậy, mọi văn phạm tuyến tính phải đều sinh ra một tập hợp chính quy.

Bây giờ, giả sử G(V, T, P, S) là một văn phạm tuyến tính trái. Đặt văn phạm G'(V, T, P', S) với P' chứa các luật sinh của P có vế phải đảo ngược, nghĩa là :

P' = { A A R P }

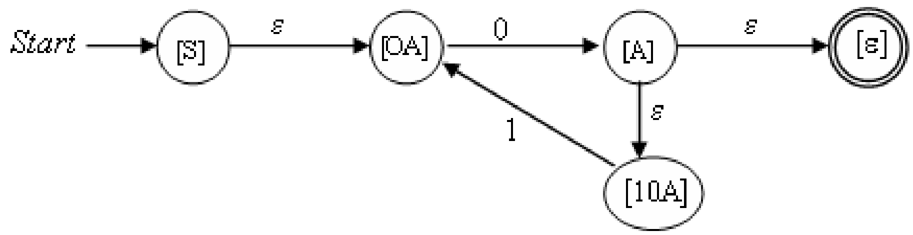
Nếu ta đảo ngược chuỗi vế phải các luật sinh trong một văn phạm tuyến tính trái, ta có văn phạm tuyến tính phải, và ngược lại. Do đó, hiển nhiên chúng ta có G' là một văn phạm tuyến tính phải, và cũng dễ dàng để chỉ ra rằng L(G') = L(G)R. Theo chứng minh trên, ta có L(G') là một tập chính quy. Mà thông thường một tập chính quy cũng vẫn còn giữ nguyên tính chất khi áp dụng phép đảo ngược nên L(G')R = L(G) cũng là một tập chính quy.

Vậy, mọi văn phạm tuyến tính trái hay phải đều sinh ra một tập hợp chính quy.

Thí dụ 4.3 : NFA được xây dựng từ Định lý 4.1 cho văn phạm tuyến tính phải (1) ở thí dụ 4.2 có dạng như hình 4.1 sau :

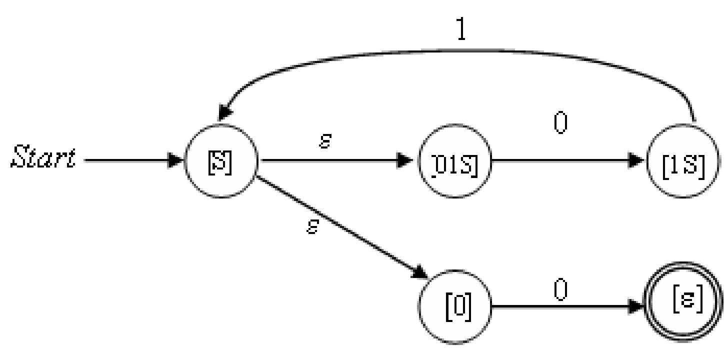
Xét văn phạm tuyến tính trái (2) ở thí dụ 4.2, nếu đảo ngược các vế phải luật sinh, ta có:

S 01S 0

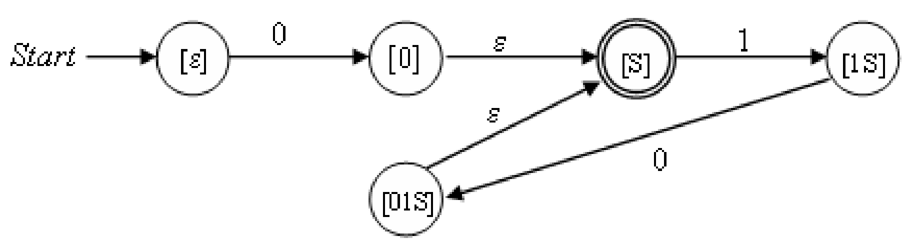


Hình 4.1 - NFA chấp nhận ngôn ngữ 0(10)*

Áp dụng các bước xây dựng NFA cho văn phạm này theo Định lý 4.1, ta có sơ đồ chuyển như Hình 4.2 (a). Nếu chúng ta đảo ngược các cạnh của NFA này và chuyển đổi vị trí các trạng thái bắt đầu và kết thúc, chúng ta sẽ có một NFA khác chấp nhận ngôn ngữ 0(10)*



Hình (a)



Hình (b)

Hình 4.2 - Xây dựng NFA cho 0(10)* từ văn phạm tuyến tính trái

ĐỊNH LÝ 4.2 : Nếu L là một tập hợp chính quy, thì L được sinh từ một văn phạm tuyến tính trái hoặc một văn phạm tuyến tính phải nào đó.

Chứng minh

Đặt L = L(M) với DFA M(Q, , , q0, F).

Trước hết, ta giả sử rằng trạng thái q0 không phải là trạng thái kết thúc. Kế tiếp, ta đặt L = L(G) với văn phạm tuyến tính phải G(V, , P, q0), trong đó P chứa các luật sinh dạng p   aq nếu (p, a) = q và luật sinh dạng p   a nếu (p, a) là một trạng thái kết thúc. Rõ ràng (p, w) = q khi và chỉ khi có chuỗi dẫn xuất p * wq. Nếu wa được chấp nhận bởi M, ta đặt (q0, w) = p, suy ra dẫn xuất q0 * wq. Tương tự, nếu (p, a) là trạng thái kết thúc, vì p   a là một luật sinh, nên q0 * wa. Ngược lại, đặt q0 * x. Ta có x = wa và q0 * wq   wa với mọi p. Và vì (q0, w) = p và (p, a) là trạng thái kết thúc nên do đó x   L(M). Hay nói cách khác : L(M) = L(G) = L.

Bây giờ, xét q0   F, vì thế chuỗi rỗng   thuộc L. Lưu ý rằng văn phạm G vừa định nghĩa ở trên chỉ sinh ra ngôn ngữ L – { }. Chúng ta có thể sửa đổi G bằng cách thêm vào một ký hiệu bắt đầu S mới với luật sinh S   q0   . Văn phạm thu được vẫn có dạng tuyến tính phải và phát sinh ngôn ngữ L.

Để phát sinh một văn phạm tuyến tính trái cho L, ta bắt đầu với một NFA cho LR và sau đó đảo ngược chuỗi vế phải cho tất cả mọi luật sinh của văn phạm tuyến tính phải vừa thu được.

Thí dụ 4.4 : Trong Hình 4.3 ta thấy sơ đồ chuyển DFA cho 0(10)*

Văn phạm tuyến tính phải sinh từ DFA này có dạng :

A   0B 1D 0

B   0D 1C

C   0B 1D 0

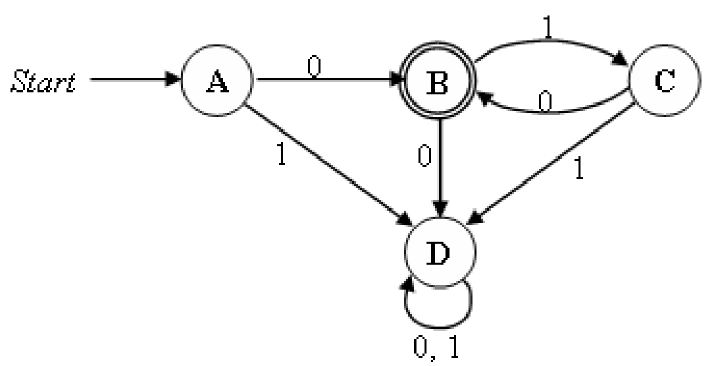
D   0D 1D

Trong văn phạm trên, biến D không có ích nên ta có thể loại bỏ D và tất cả các luật sinh liên quan tới D, rút gọn văn phạm thành :

A   0B 0

B   1C

C   0B 0



Hình 4.3 - DFA cho 0(10)*

MỘT SỐ TÍNH CHẤT CỦA TẬP HỢP CHÍNH QUY

Một câu hỏi khá quan trọng được đặt ra là: Cho ngôn ngữ L với một số tính chất đặc tả nào đó, liệu L có phải là tập chính quy không ? Phần này cung cấp một số lý thuyết giúp trả lời câu hỏi này.

Bổ đề bơm cho tập hợp chính quy

Một trong những nguyên lý hiệu quả là "Bổ đề bơm", đây là một công cụ mạnh giúp chứng minh các ngôn ngữ không là chính quy. Đồng thời, nó cũng thực sự hữu ích trong việc phát triển các giải thuật liên quan đến các ôtômát, chẳng hạn như một ngôn ngữ được chấp nhận bởi một FA cho trước là hữu hạn hay vô hạn ?

BỔ ĐỀ 4.1: (BỔ ĐỀ BƠM)

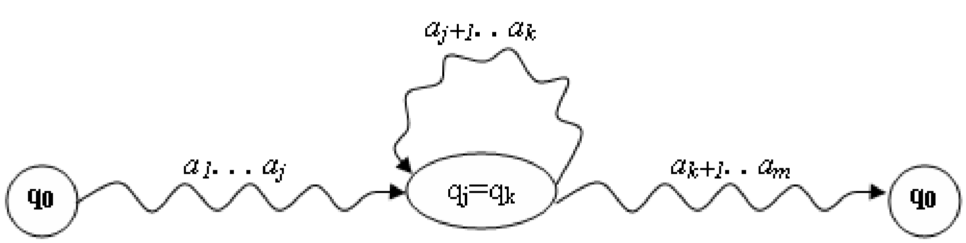
Nếu L là tập hợp chính quy thì có tồn tại hằng số n sao cho nếu z là một từ bất kỳ thuộc L và | z | n, ta có thể viết z = uvw với | uv | n, | v | 1 và i   0, ta có uviw   L.

Hơn nữa n không lớn hơn số trạng thái của FA nhỏ nhất chấp nhận L.

Chứng minh

Nếu một ngôn ngữ L là ngôn ngữ chính quy thì nó sẽ được chấp nhận bởi một DFA M (Q, , , q0, F) với n trạng thái.

Xét chuỗi nhập z có m ký hiệu được cho như trong bổ đề, vậy z = a1a2 ... am, m   n, và với mỗi i = 1, 2, ..., m , ta đặt (q0, a1a2...ai) = qi. Do m   n nên cần phải có ít nhất n+1 trạng thái trên đường đi của ôtômát chấp nhận chuỗi z. Trong n+1 trạng thái này phải có hai trạng thái trùng nhau vì ôtômát M chỉ có n trạng thái phân biệt, tức là có hai số nguyên j và k sao cho 0   j < k   n thỏa mãn qj = qk. Đường đi nhãn a1a2 ... am trong sơ đồ chuyển của M có dạng như sau:



Hình 4.4 - Đường đi trong sơ đồ chuyển của DFA M

Vì j < k nên chuỗi aj+1...ak có độ dài ít nhất bằng 1 và vì k   n nên độ dài đó không thể lớn hơn n.

Nếu qm là một trạng thái trong F, nghĩa là chuỗi a1a2...am thuộc L(M), thì chuỗi a1a2...aj ak+1ak+2...am cũng thuộc L(M) vì có một đường dẫn từ

q0 đến qm ngang qua qj nhưng không qua vòng lặp nhãn aj+1... ak. Một cách hình thức, ta có :

d(q0, a1a2...aj ak+1ak+2...am) = d (d(q0, a1a2...aj), ak+1ak+2...am)

= d (qj, ak+1ak+2...am)

= d (qk, ak+1ak+2...am)

= qm

Vòng lặp trong hình trên có thể được lặp lại nhiều lần - thực tế, số lần muốn lặp là tùy ý, do đó chuỗi a1...aj (aj+1...ak)i ak+1...am   L(M), i   0. Điều ta muốn chứng tỏ ở đây là với một chuỗi có độ dài bất kỳ được chấp nhận bởi một FA, ta có thể tìm được một chuỗi con gần với chuỗi ban đầu mà có thể "bơm" - lặp một số lần tùy ý - sao cho chuỗi mới thu được cũng được chấp nhận bởi FA.

Đặt u = a1...aj, v = aj+1...ak và w = ak+1...am.

Ta có điều phải chứng minh.

Ứng dụng của bổ đề bơm

Bổ đề bơm rất có hiệu quả trong việc chứng tỏ một tập hợp không là tập hợp chính quy. Phương pháp chung để ứng dụng nó dùng phương pháp chứng minh “phản chứng” theo dạng sau :

1. Chọn ngôn ngữ mà bạn cần chứng tỏ đó không là ngôn ngữ chính quy.
2. Chọn hằng số n, hằng số được đề cập đến trong bổ đề bơm.
3. Chọn chuỗi z thuộc L. Chuỗi z phải phụ thuộc nghiêm ngặt vào hằng số n đã chọn ở bước 2.
4. Giả thiết phân chuỗi z thành các chuỗi con u, v, w theo ràng buộc | uv | n và | v | 1
5. Mâu thuẫn sẽ phát sinh theo bổ đề bơm bằng cách chỉ ra với u, v và w xác định theo giả thiết, có tồn tại một số i mà ở đó uviw   L. Từ

đó có thể kết luận rằng L không là ngôn ngữ chính quy. Chọn lựa giá trị cho i có thể phụ thuộc vào n, u, v và w.

Ta có thể phát biểu một cách hình thức nội dung của bổ đề bơm như sau :

(L)(n)(z)[z thuộc L và | z | n ta có

(u, v, w)(z = uvw, | uv | n, | v | 1 và (i)(uviw thuộc L))]

Thí dụ 4.5 : Chứng minh tập hợp L = { 0i2 i là số nguyên, i 1} (L chứa tất cả các chuỗi số 0 có độ dài là một số chính phương) là tập không chính quy.

Chứng minh

Giả sử L là tập chính quy và tồn tại một số n như trong bổ đề bơm.

Xét từ z =0n2. Theo bổ đề bơm, từ z có thể viết là z = uvw với 1 | v | n và uviw L, i 0. Trường hợp cụ thể, xét i = 2 : ta phải có uv2w L.

Mặt khác : n2 < | uv2w | n2 + n < (n+1)2.

Do n2 và (n+1)2 là 2 số chính phương liên tiếp nên | uv2w | không thể bằng một số chính phương, vậy uv2w L.

Điều này dẫn đến sự mâu thuẫn, vậy giả thiết ban đầu là sai. Suy ra L không là tập chính quy.

Câu hỏi :

?

Hãy tự liên hệ một số tập ngôn ngữ khác mà bạn nghĩ chúng không thuộc lớp

ngôn ngữ chính quy vì không thể thỏa mãn các tính chất của Bổ đề bơm ?

Tính chất đóng của tập hợp chính quy

Có nhiều phép toán trên ngôn ngữ chuyên sử dụng cho tập hợp chính quy, mà cho phép khi áp dụng chúng vào tập hợp chính quy thì vẫn giữ được các tính chất của tập chính quy. Nếu một lớp ngôn ngữ nào đó "đóng" với một phép toán cụ thể, ta gọi đó là tính chất đóng của lớp ngôn ngữ này.

ĐỊNH LÝ 4.3 : Tập hợp chính quy đóng với các phép toán: hợp, nối kết và bao đóng Kleen.

Chứng minh

Hiển nhiên từ định nghĩa của biểu thức chính quy.

ĐỊNH LÝ 4.4 : Tập hợp chính quy đóng với phép lấy phần bù. Tức là, nếu L là tập chính quy và L * thì * - L là tập chính quy.

Chứng minh

Gọi L là L(M) cho DFA M (Q, 1, , q0, F) và L *.

Trước hết, ta giả sử 1 = vì nếu có ký hiệu thuộc 1 mà không thuộc thì ta có thể bỏ các phép chuyển trong M liên quan tới các ký hiệu đó. Do L * nên việc xóa như vậy không ảnh hưởng tới M. Nếu có ký hiệu thuộc nhưng không thuộc 1 thì các ký hiệu này không xuất hiện trong L. Ta thiết kế thêm một trạng thái "chết" d trong M sao cho (d, a) = d, a và (q, a) = d, q Q và a - 1.

Bây giờ, để chấp nhận * - L, ta hoàn thiện các trạng thái kết thúc của M. Nghĩa là, đặt M’ = (Q, , , q0, Q - F). Ta có M’ chấp nhận từ w nếu (q0,w) Q - F, suy ra w Î * - L.

ĐỊNH LÝ 4.5: Tập hợp chính quy đóng với phép giao

Chứng minh

Do ta có công thức biến đổi :

$L_1 \quad L_2 = L_1 \quad L_2$

Nên theo các định lý trên, suy ra được tập L1   L2 là tập chính quy.

các GIẢI THUẬT xác định TẬP hỢp CHÍNH QUY

Một vấn đề khác, cũng rất cần thiết là xác định các giải thuật giúp giải đáp nhiều câu hỏi liên quan đến tập hợp chính quy, chẳng hạn như : Một ngôn ngữ cho trước là rỗng, hữu hạn hay vô hạn ? Ngôn ngữ chính quy có tương đương với ngôn ngữ nào khác không ? ... Để xác định các giải thuật này, trước hết cần giả sử mỗi tập chính quy thì được biểu diễn bởi một ôtômát hữu hạn. Như đã biết, biểu thức chính quy dùng đặc tả cho tập hợp chính quy, do đó chỉ cần cung cấp thêm một cơ chế dịch từ dạng biểu thức này sang dạng ôtômát hữu hạn. Một số định lý sau có thể xem là nền tảng cho việc chuyển đổi này.

ĐỊNH LÝ 4.6: Tập hợp các chuỗi được chấp nhận bởi ôtômát M có n trạng thái là:

1) Không rỗng nếu và chỉ nếu ôtômát chấp nhận một chuỗi có độ dài < n.

2) Vô hạn nếu và chỉ nếu ôtômát chấp nhận một chuỗi có độ dài l với n l < 2n.

Chứng minh

1) Phần "nếu " là hiển nhiên.

Ta chứng minh "chỉ nếu": Giả sử M chấp nhận một tập không rỗng. Gọi w là chuỗi ngắn nhất được chấp nhận bởi M. Theo bổ đề bơm, ta có | w | < n vì nếu w là chuỗi ngắn nhất và | w | n thì ta có thể viết w = uvy, và uy là chuỗi ngắn hơn trong L hay | uy | < | w | Mâu thuẫn.

2) Nếu w   L và n | w | < 2n thì theo bổ đề bơm ta có w = w1w2w3 và w1w2 i w3   L với mọi i   0, suy ra L(M) vô hạn .

Ngược lại, nếu L(M) vô hạn thì tồn tại w   L(M) sao cho | w | n. Nếu | w |< 2n thì xem như đã chứng minh xong. Nếu không có chuỗi nào có độ dài nằm giữa n và 2n-1 thì gọi w là chuỗi có độ dài ít nhất là 2n nhưng ngắn hơn mọi chuỗi trong L(M), nghĩa là | w | 2n. Một lần nữa, cũng theo bổ đề bơm, ta có thể biểu diễn w = w1w2w3, trong đó 1 | w2 | n và w1w3   L(M). Ta có hoặc w không phải là chuỗi ngắn nhất có độ dài 2n, hoặc là n | w1w3 | 2n-1   Mâu thuẫn. Vậy có tồn tại chuỗi có độ dài l sao cho n   l < 2n.

ĐỊNH LÝ 4.7 : Có giải thuật để xác định hai ôtômát tương đương (chấp nhận cùng một ngôn ngữ).

Chứng minh

Đặt M1, M2 là hai ôtômát chấp nhận L1, L2.

Theo các định lý 4.3, 4.4, 4.5, ta có (L_1 L_2)   (L_1 L_2) được chấp nhận bởi ôtômát M3 nào đó. Dễ thấy M3 chấp nhận một chuỗi nếu và chỉ nếu L1   L2. Theo định lý 4.6, ta thấy có giải thuật để xác định xem liệu L1 = L2 hay không.

Tổng kết chương IV: Qua chương này, chúng ta có thể thấy rõ hơn các tính chất của lớp ngôn ngữ chính quy và cách xác định chúng bằng một số giải thuật. Mối liên quan giữ hai cơ chế đoán nhận ngôn ngữ (ôtômát hữu hạn) và phát sinh ngôn ngữ (văn phạm) cũng đã được thiết lập và chứng minh rõ ràng. Đây là lớp ngôn ngữ nhỏ nhất theo sự phân cấp của Noam Chomsky. Trong những chương tiếp theo, chúng ta sẽ khảo sát những lớp ngôn ngữ rộng lớn hơn chứa cả ngôn ngữ chính quy trong nó.

BÀI TẬP CHƯƠNG IV

4.1. Xây dựng văn phạm tuyến tính trái và tuyến tính phải cho các ngôn ngữ sau :

a) (0 + 1)*00(0 + 1)*

b) 0*(1(0 + 1))*

c) (((01 + 10)*11)*00)*

4.2. Xây dựng văn phạm chính quy sinh ra các ngôn ngữ trên bộ chữ cái = {0,1} như sau :

a) Tập các chuỗi có chứa 3 con số 0 liên tiếp.

b) Tập các chuỗi kết thúc bằng 2 con số 0.

4.3. Xây dựng văn phạm chính quy sinh ra các ngôn ngữ sau :

a) { w   w   (0 + 1)* }

b) { am bn   m, n > 0 }

4.4. Chứng tỏ rằng ngôn ngữ L = {0n1n   n là số nguyên dương} không chính qui.

4.5. Ngôn ngữ nào trong các ngôn ngữ sau không là ngôn ngữ chính qui? Chứng minh câu trả lời:

1. L = {02n   n là số nguyên dương }
2. L = {0n1m0 n+m   m, n là số nguyên dương}
3. L = {0n   n là số nguyên tố }

Máy Turing
Nội dung chính : Trong chương này, ta sẽ xét thêm một loại máy trừu tượng khác - máy Turing (TM - Turing Machines). Chúng có khả năng đoán nhận được lớp ngôn ngữ lớn hơn lớp ngôn ngữ phi ngữ cảnh. Đây còn là một mô hình của sự tính toán, mô hình của các thủ tục hiệu quả, là nền tảng cho quá trình xử lý của máy tính hiện đại, được giới thiệu bởi Alan Turing vào năm 1936. Nhờ đó, các khái niệm về "sự tính được", "sự giải được" được xác định một cách rõ ràng trên cơ sở sự xuất hiện của một số hàm không tính được, các bài toán không giải được. Mục tiêu cần đạt: Cuối chương, sinh viên cần phải nắm vững: Khái niệm TM, định nghĩa và các thành phần. Các kỹ thuật thiết kế TM. Một số biến dạng TM từ mô hình chuẩn. Xây dựng TM dùng nhận dạng ngôn ngữ hoặc tính toán các hàm số nguyên đơn giản được biểu diễn trong hệ nhất phân. Các tính chất của lớp ngôn ngữ được chấp nhận bởi TM. Kiến thức cơ bản: Để tiếp thu tốt nội dung của chương này, sinh viên cần hiểu rõ cách thiết kế các hàm chuyển trạng thái trên mô hình máy tính toán; ý tưởng thiết kế một số thuật toán đơn giản trên tập hợp số, … Tài liệu tham khảo : [1] John E. Hopcroft, Jeffrey D.Ullman – Introduction to Automata Theory, Languages and Computation – Addison – Wesley Publishing Company, Inc – 1979 (Chapter 7 : Turing Machines) [2] Peter Linz – An Introduction to Formal Languages and Automata – D.C. Heath and Company – 1990. [3] David Barker-Plummer - Stanford Encyclopedia of Philosophy – Turing Machines: http://plato.stanford.edu/entries/turing-machine/ [4] Turing Machinesimplemented in JavaScript: http://www.turing.org.uk/turing/scrapbook/tmjava.html [5] By Jon Barwise and John Etchemendy -Turing Machines: http://www-csli.stanford.edu/hp/Turing1.html

MÔ HÌNH MÁY TURING (TM)

Một mô hình hình thức cho một thủ tục hiệu quả sẽ có những đặc tính cụ thể. Đầu tiên, mỗi thủ tục sẽ được mô tả một cách hữu hạn. Tiếp đó, thủ tục sẽ được phân thành một số bước độc lập, mà mỗi bước thực thi một vấn đề. Nguyên tắc này cũng được hình thức trong mô hình máy Turing.

Máy Turing có một băng nhớ, dùng để ghi mọi loại dữ liệu (dữ liệu nhập, dữ liệu dùng cho việc điều khiển tương tự như một chương trình máy tính và các kết quả trung gian khi làm việc). Với một bộ điều khiển chứa một số hữu hạn trạng thái, TM cũng như các ôtômát khác, làm việc theo lối "ngắt quãng" theo từng bước chuyển.

Mô tả TM

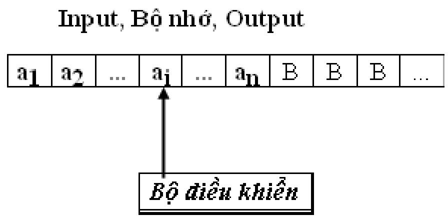
Máy Turing có rất nhiều dạng đồng khả năng, nghĩa là có nhiều mô hình và định nghĩa khác nhau cho máy Turing nhưng tất cả chúng đều tương đương nhau. Song, nói chung mô hình cơ bản của một máy Turing gồm :

- Một bộ điều khiển hữu hạn.

- Một băng được chia thành các ô.

- Một đầu đọc-viết, mỗi lần đọc có thể duyệt qua một ô trên băng để đọc hay viết ký hiệu.

Mỗi ô có thể giữ được một ký hiệu trong số hữu hạn các ký hiệu băng (các ký hiệu được phép viết trên băng). Khởi đầu xem như n ô bên trái của băng (n 0) giữ chuỗi nhập (input), chuỗi nhập là một chuỗi các ký tự được chọn từ một tập hợp con của tập hợp các ký hiệu băng, tập hợp con này gọi là tập các ký hiệu nhập. Phần còn lại của băng coi như có vô hạn khoảng trống, ký hiệu B (Blank), B là một ký hiệu đặc biệt của băng nhưng không phải là ký hiệu nhập.



Hình 7.1 - Mô tả một TM

Mỗi bước chuyển của máy Turing, phụ thuộc vào ký hiệu do đầu đọc đọc được trên băng và trạng thái của bộ điều khiển, máy sẽ thực hiện các bước sau :

1) Chuyển trạng thái

2) In một ký hiệu trên băng tại ô đang duyệt (nghĩa là thay ký hiệu đọc được trên băng bằng ký hiệu nào đó)

3) Dịch chuyển đầu đọc-viết (sang trái (L), sang phải (R) hoặc đứng yên( ))

Câu hỏi :

?So sánh cơ chế máy Turing với hai dạng ôtômát đã khảo sát trong các chương trước (ôtômát hữu hạn FA và ôtômát đẩy xuống PDA) ? Nêu những điểm khác biệt quan trọng trong nguyên tắc nhận dạng ngôn ngữ ?

Định nghĩa

Một cách hình thức, ta định nghĩa một máy Turing (TM) như sau :

Định nghĩa: TM là một hệ thống M (Q, ∑, , , q0, B, F), trong đó:

. Q : tập hữu hạn các trạng thái.

. ∑: bộ ký hiệu nhập.

. : tập hữu hạn các ký tự được phép viết trên băng.

. B : ký hiệu thuộc dùng chỉ khoảng trống trên băng (Blank).

. : hàm chuyển ánh xạ : Q Q {L, R, }

(có thể không xác định với một vài đối số)

. q0 Q là trạng thái bắt đầu

. F Q là tập các trạng thái kết thúc

Hình thái TM (Instantaneous description - ID)

Một hình thái của máy Turing M được cho bởi 1 q 2, trong đó q là trạng thái hiện hành của M; 1 2 * là nội dung của băng tính từ đầu băng cho tới ký hiệu khác Blank bên phải nhất của băng. Giả sử Q và rời nhau: đầu đọc đang đọc ký hiệu bên trái nhất của 2 hoặc nếu 2 = thì đầu đọc đọc Blank.

Hàm chuyển

Ta định nghĩa một phép chuyển trạng thái của TM như sau :

Đặt X1X2 ... Xi-1 q Xi ... Xn là một ID.

+ Giả sử (q, Xi) = (p, Y, L), trong đó:

- Nếu i - 1 = n thì Xi là B.

- Nếu i =1 thì không có ID kế tiếp, nghĩa là đầu đọc không được phép vượt qua cận trái của băng.

- Nếu i > 1 ta viết :

X1X2 ... Xi-1 q Xi ... Xn ⊢M X1X2 ... Xi-2 p Xi-1Y Xi+1 ... Xn

+ Tương tự (q, Xi) = (p, Y, R) thì ta viết :

X1X2 ... Xi-1 q Xi ... Xn ⊢M X1X2 ... Xi-1 Yp Xi+1 ... Xn

+ Tương tự (q, Xi) = (p, Y,) thì ta viết :

X1X2 ... Xi-1 q Xi ... Xn ⊢M X1X2 ... Xi-1 pY Xi+1 ... Xn

Chú ý rằng nếu i - 1 = n thì chuỗi Xi ... Xn là rỗng và vế phải dài hơn vế trái, nghĩa là TM M mở rộng chuỗi ký hiệu trên băng.

Nếu hai ID được quan hệ nhau bởi ⊢M thì ta nói ID thứ hai là kết quả của ID thứ nhất bằng một lần chuyển, một bước áp dụng hàm chuyển (hoặc nói cái thứ hai thu được từ cái thứ nhất bằng một lần chuyển). Nếu một ID thu được từ ID khác bằng một số lần chuyển (có thể bằng 0) thì ta ký hiệu quan hệ là ⊢M* . Ta cũng có thể bỏ đi ký hiệu M trong cách viết các quan hệ trên nếu không có nhầm lẫn.

Ngôn ngữ được chấp nhận bởi TM

Ký hiệu L(M): tập hợp các chuỗi trong * là nguyên nhân đưa TM M đi vào trạng thái kết thúc khi đã thực hiện việc thay thế từ bên trái các ký hiệu trên băng của M với trạng thái bắt đầu q0. Một cách hình thức, ta định nghĩa tập hợp ngôn ngữ được chấp nhận bởi TM M (Q, ∑, , , q0, B, F) là tập

L(M) = { w w   * và q0 w ⊢M* 1 p 2 với p F còn 1 2   *}

Cho TM nhận diện một ngôn ngữ L là cho lần lượt các từ của L vào TM xem TM có chấp nhận từ đó không. TM sẽ dừng và đi vào một trong những trạng thái kết thúc F (không có phép chuyển kế tiếp) khi từ được chấp nhận, nhưng nếu TM không chấp nhận một từ nào đó thì TM có thể ngừng ở một trạng thái F hoặc cũng có thể nó chạy mãi mà không dừng lại.

Thí dụ 7.1 : Thiết kế TM chấp nhận ngôn ngữ L = { 0n1n n 1}

Khởi đầu TM chứa 0n1n bên trái nhất trên băng sau đó là vô hạn khoảng trống Blank. TM lặp lại quá trình sau:

- M thay 0 bên trái nhất bằng X rồi chuyển sang phải tới 1 trái nhất, TM thay 1 này bằng Y rồi dịch chuyển về bên trái cho tới khi gặp X phải nhất nó chuyển sang phải một ô (tới 0 trái nhất) rồi tiếp tục lặp một chu trình mới.

- Nếu trong khi dịch chuyển sang phải để tìm 1 mà TM gặp Blank thì TM dừng và không chấp nhận input. Tương tự, khi TM đã thay hết 0 bằng X và kiểm tra còn 1 trên băng thì TM cũng dừng và không chấp nhận input.

- TM chấp nhận input nếu như cũng không còn ký hiệu 1 nào nữa trên băng.

Đặt TM M (Q, ∑, , , q0, B, F) với các thành phần :

Q = {q0, q1, q2, q3, q4}; ∑= {0, 1};   = {0, 1, X, Y, B} và F = {q4}.

Ta có thể hình dung mỗi trạng thái là một câu lệnh hoặc một nhóm các câu lệnh trong chương trình. Trạng thái q0 là trạng thái khởi đầu và nó làm cho ký hiệu 0 bên trái nhất thay bằng X. Trạng thái q1 được dùng để tiến sang phải bỏ qua các số 0 và Y để tìm 1 bên trái nhất. Nếu M tìm thấy 1 nó thay 1 bằng Y rồi đi vào trạng thái q2. Trạng thái q2 đưa M tiến sang trái cho tới X đầu tiên và đi vào trạng thái q0, dịch chuyển sang phải để tới 0 bên trái nhất và tiếp tục một chu trình mới. Khi M tiến sang phải trong trạng thái q1, nếu B hoặc X được tìm thấy trước 1 thì input bị loại bỏ (không chấp nhận) vì có chứa nhiều ký hiệu 0 hơn 1 hoặc input không có dạng 0*1* .

Trạng thái q0 còn có vai trò khác. Nếu trạng thái q2 tìm thấy X bên phải nhất và ngay sau đó là Y thì các số 0 đã được xét hết, do đó ở trạng thái bắt đầu một chu trình mới q0 không tìm thấy ký hiệu 0 nào để thay thành X mà chỉ gặp Y thì TM đi vào trạng thái q3 duyệt qua các Y để kiểm tra có hay không có ký hiệu 1 còn lại. Nếu theo ngay sau các Y là B, nghĩa là trên băng nhập không còn ký hiệu 1 nào nữa thì TM sẽ đi vào q4 (trạng thái kết thúc) để chấp nhận input. Ngược lại input bị loại bỏ.

Hàm chuyển   được cho trong bảng sau :

			Ký hiệu		
Trạng thái	0	1	X	Y	B
q0	(q1, X, R)	-	-	(q3, Y, R)	-
q1	(q1, 0, R)	(q2, Y, L)	-	(q1, Y, R)	-
q2	(q2, 0, L)	-	(q0, X, R)	(q2, Y, L)	-
q3	-	-	-	(q3, Y, R)	(q4, B,)
q4	-	-	-	-	-

Các phép chuyển hình thái của TM M trên input 0011 :

q00011 ⊢ Xq1011 ⊢ X0q111 ⊢ X q20Y1 ⊢ q2X0Y1 ⊢ X q00Y1 ⊢ XXq1Y1 ⊢ XXY q11 ⊢ XX q2YY ⊢ X q2XYY ⊢ XX q0YY ⊢ XXYq3Y ⊢ XXYYq3 ⊢ XXYYq4

Nhận xét: Như vậy, ta có thể dễ dàng thấy, TM khác với một ôtômát hữu hạn ở chỗ đầu đọc-viết có thể dịch chuyển tự do trên băng, không những đọc mà còn có khả năng viết trên băng và vùng làm việc còn có thể mở rộng theo yêu cầu phát sinh. TM khác với ôtômát đẩy xuống ở chỗ nó không dùng thêm Stack như một bộ giữ nhớ mà viết các ký hiệu cần ghi nhớ ngay trên băng.

NGÔN NGỮ VÀ "HÀM TÍNH ĐƯỢC"

Ngôn ngữ được chấp nhận bởi một máy Turing được gọi là ngôn ngữ đệ qui liệt kê - recursively enumerable (r.e). Đó là một lớp ngôn ngữ rất rộng, nó thực sự chứa ngôn ngữ phi ngữ cảnh CFL và một số ngôn ngữ mà không thể xác định các thành phần một cách máy móc. Nếu L(M) là một ngôn ngữ như vậy thì bất kỳ một máy Turing nào nhận diện L(M) cũng sẽ không dừng trên một số input không thuộc L(M). Nhưng nếu một chuỗi w   L(M) thì chắc chắn TM dừng, tuy nhiên TM sẽ chạy bao lâu trên input thì chúng ta không thể biết được và ta cũng không biết chắc chắn liệu TM có dừng lại hay không. Một cách thuận lợi và có ý nghĩa hơn là xét một lớp con của lớp ngôn ngữ đệ qui liệt kê, trong đó mọi ngôn ngữ đều được chấp nhận bởi ít nhất một máy Turing dừng trên mọi input. Lớp ngôn ngữ này gọi là lớp ngôn ngữ đệ qui - recursive sets.

Máy Turing như là một máy tính hàm số nguyên

Máy Turing cũng có thể được xem như là một máy tính của các hàm số nguyên (đi từ tập số nguyên đến tập số nguyên). Mỗi số nguyên ta viết dưới dạng số trong hệ nhất phân (unary), tức là với một số i   0 ta viết thành chuỗi 0i (gồm i chữ số 0). Nếu hàm f có k đối số i1, i2, ..., ik thì ta viết lần lượt các số nguyên này trên băng của TM ngăn cách nhau bởi 1, nghĩa là input có dạng 0i110i21 ... 10ik. Nếu TM dừng (chấp nhận hoặc không chấp nhận input) với băng 0m thì ta nói f (i1, i2, ..., ik) = m.

Chú ý rằng ta cũng có thể tính được hàm chỉ có một đối số. Nếu f xác định với mọi bộ đối số i1, i2, ..., ik thì ta gọi f là hàm đệ qui toàn bộ. Một hàm f tính được bởi máy Turing ta gọi là hàm đệ qui bộ phận. Hàm đệ qui bộ phận tương tự như ngôn ngữ đệ qui liệt kê bởi vì nó tính được bởi máy Turing nhưng có thể không dừng với một số đối số nào đó. Hàm đệ qui toàn bộ tương tự như ngôn ngữ đệ qui vì TM sẽ dừng trên mọi input.

Thí dụ 7.2 :Thiết kế máy Turing tính toán phép trừ riêng

Ta định nghĩa phép trừ riêng (proper subtraction) như sau :

f(m, n) = m\ n =m - n nếu m   n

1. nếu m < n

. Input : 0m10n

. Output : 0 m\ n

M lặp lại việc thay thế lần lượt từng số 0 ở đầu băng bằng B rồi tiến sang phải, ra sau 1 tìm 0 và thay 0 này bằng 1. M lại chuyển sang trái cho đến khi gặp B đầu tiên thì dừng lại, trở về trạng thái bắt đầu và tiếp tục vòng lặp như trên. M dừng nếu :

i) Khi sang phải tìm 0 bên phải, M gặp B. Lúc này M đã thay n số 0 bên phải chuỗi input 0m10n thành 1 và n + 1 số 0 bên trái thành B, trường hợp này xảy ra khi trong chuỗi input có m > n. Do vậy M phải thay lại tất cả n + 1 số 1 sau thành B, và sau đó dịch trái thay trả lại một B về thành 0, cuối cùng trên băng còn lại kết quả phép trừ là m - n số 0.

ii) Khi bắt đầu một vòng lặp mới, M không tìm thấy 0 để đổi thành B, lúc này m số 0 đầu đã bị đổi thành B, trường hợp này xảy ra khi n   m. Khi đó, M thay tất cả các số 1 và 0 trên băng thành B để cho kết quả phép trừ là 0 (biểu diễn gồm toàn ký hiệu B trong hệ nhất phân).

Ta xây dựng TM như sau: M ({q0, q1, ..., q6}, {0, 1}, {0, 1, B}, , q0, B, {q6})

M sẽ bắt đầu bằng 0m10n trên băng và kết thúc với 0m\ n trên băng. Các phép chuyển trạng thái được định nghĩa như sau :

1) d(q0, 0) = (q1, B, R)

M thay 0 đầu băng bởi B.

2) d(q1, 0) = (q1, 0, R)

d(q1, 1) = (q2, 1, R)

M di chuyển sang phải qua 0 tìm 1.

3) d(q2, 1) = (q2, 1, R)

d(q2, 0) = (q3, 1, L)

M di chuyển sang phải vượt qua 1 đến khi gặp 0, đổi 0 thành 1.

4) d(q3, 0) = (q3, 0, L)

d(q3, 1) = (q3, 1, L)

d(q3, B) = (q0, B, R)

M dịch trái tới khi gặp B, trở về trạng thái q0 và bắt đầu một vòng lặp mới.

5) d(q2, B) = (q4, B, L)

d(q4, 1) = (q4, B, L)

d(q4, 0) = (q4, 0, L)

d(q4, B) = (q6, 0,)

Nếu ở trạng thái q2 sang phải tìm 0 để thay thành 1 nhưng chỉ gặp B thì ta xét trường hợp kết thúc i) ở trên: TM đi vào trạng thái q4 và chuyển sang trái đổi tất cả 1 thành B cho tới khi gặp một B bên trái đầu tiên. B này sẽ được thay lại thành 0 rồi M đi vào trạng thái kết thúc q6 và dừng.

6) d(q0, 1) = (q5, B, R)

d(q5, 0) = (q5, B, R)

d(q5, 1) = (q5, B, R)

d(q5, B) = (q6, B,)

Nếu ở trạng thái bắt đầu vòng lặp mới q0 gặp 1 thay vì gặp 0, thì khối các số 0 bên trái đã xét hết, đây là trường hợp kết thúc ii) nêu trên: TM sẽ đi vào trạng thái q5, xoá phần còn lại của băng rồi đi vào trạng thái kết thúc q6 và dừng.

Chẳng hạn TM tính toán phép trừ 2\1 (tức input 0010) như sau :

q00010 ⊢ B q1010 ⊢ B0q110 ⊢ B01q20 ⊢ B0q311 ⊢ Bq3011 ⊢ q3B011 ⊢ Bq0011 ⊢ BBq111 ⊢ BB1q21 ⊢ BB11q2 ⊢ BB1q41 ⊢ BBq41 ⊢ Bq4 ⊢ Bq60

Nếu cho TM tính toán 1\2 (tức input 0100) :

q00100 ⊢ Bq1100 ⊢ B1q200 ⊢ Bq3110 ⊢ q3B110 ⊢ Bq0110 ⊢ BBq510 ⊢ BBBq50 ⊢ BBBBq5 ⊢ BBBBq6

CÁC KỸ THUẬT XÂY DỰNG MÁY TURING

Việc xây dựng máy Turing bằng cách viết (liệt kê) tất cả các hàm chuyển của nó trên băng nhập có thể là một công việc đơn điệu. Để mô tả đầy đủ cách xây dựng máy Turing, ta cần một vài công cụ "cấp cao" hơn. Phần này sẽ trình bày một số công cụ tổng quát :

Lưu trữ trong bộ điều khiển (Storage in the finite control)

Bộ điều khiển có thể dùng để lưu trữ một lượng hữu hạn thông tin. Để làm như thế, ta viết mỗi trạng thái như là một cặp các phần tử: một thành phần để điều khiển, thành phần kia lưu giữ 1 ký hiệu. Chú ý rằng, đây chỉ là một cách mở rộng trên khái niệm chứ không thay đổi định nghĩa máy Turing.

Thí dụ 7.3 :Xét máy Turing M nhận vào ký hiệu đầu tiên trên chuỗi nhập (viết trên bộ chữ cái {0, 1}), lưu trữ vào bộ điều khiển và kiểm tra rằng ký hiệu này không có xuất hiện ở vị trí nào khác trên chuỗi nữa hay không ?.

Ta xây dựng TM M (Q, {0, 1}, {0, 1, B}, , [q0, B], B, F}), trong đó tập trạng thái Q bao gồm các trạng thái dạng một cặp thành phần {q0, q1} {0, 1, B}, tức là Q gồm chứa các trạng thái [q0, 0], [q0, 1], [q0, B], [q1, 0], [q1, 1] và [q1, B]. Trong mỗi cặp này thành phần thứ nhất ghi trạng thái điều khiển, thành phần thứ hai ghi nhớ ký hiệu. Ta định nghĩa hàm chuyển như sau:

1)d([q0, B], 0) = ([q1, 0], 0, R)

d([q0, B], 1) = ([q1, 1], 1, R)

Bắt đầu từ trạng thái [q0, B], TM đọc và lưu trữ ký hiệu đầu tiên trên băng vào thành phần thứ hai trong bộ điều khiển.

2)d([q1, 0], 1) = ([q1, 0], 1, R)

d([q1, 1], 0) = ([q1, 1], 0, R)

Nếu các ký hiệu được đọc tiếp theo không giống với ký hiệu đang lưu trữ thì tiếp tục di chuyển sang phải.

3)d([q1, 0], B) = ([q1, B], 0,)

d([q1, 1], B) = ([q1, B], 0,)

M đi vào trạng thái kết thúc [q1, B] khi gặp Blank.

M sẽ đi vào trạng thái kết thúc nếu nó tiến đến gặp ký hiệu B mà không có ký hiệu nào giống với ký hiệu đầu tiên đang được lưu trữ trong bộ điều khiển. Vậy nếu M tiến đến B ở trạng thái [q1, 0] hoặc [q1, 1] thì input được chấp nhận. Ngược lại, ở trạng thái [q1, 0] và gặp 0 hoặc ở trạng thái [q1, 1] và gặp 1 thì M dừng và không chấp nhận chuỗi input vì không có hàm chuyển trạng thái để xác định các bước chuyển này.

Một cách tổng quát, ta có thể xem bộ điều khiển gồm k thành phần trong đó một thành phần giữ trạng thái điều khiển và các thành phần kia (k-1 thành phần) dùng lưu giữ thông tin.

Nhiều rãnh trên băng (Multiple tracks)

Một cách mở rộng khác, ta cũng có thể xem băng của TM được chia thành k thành phần, với k > 1 và hữu hạn. Một ký hiệu trên băng được xét là một bộ gồm k ký hiệu, mỗi ký hiệu nằm trên một rãnh.

Thí dụ 7.4 : Thiết kế TM nhận vào một số nguyên n (viết ở dạng nhị phân) và kiểm tra xem đó có phải là số nguyên tố hay không ?

Ta dùng băng 3 rãnh như hình 7.2 với nguyên tắc sau :

Số n ở dạng nhị phân được đưa vào trên rãnh 1 và được bao bởi cặp dấu Ë và $. Như vậy các ký hiệu được phép ghi trên băng là [Ë, B, B], [0, B, B], [1, B, B] và [$, B, B]. Các ký hiệu này tương ứng với Ë, 0, 1, $ khi xem chúng là ký hiệu nhập. Ký hiệu Blank là [B, B, B].

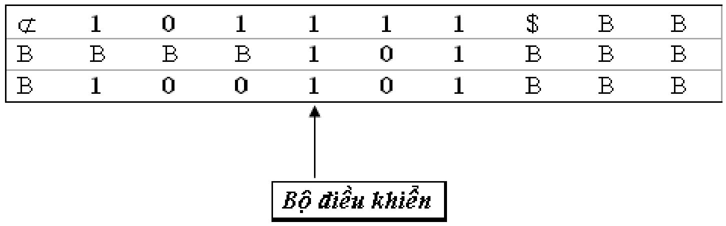
Viết số 2 dạng nhị phân trên rãnh 2 (tức 10)

Chép rãnh 1 vào rãnh 3 sau đó lấy rãnh 3 trừ rãnh 2 nhiều lần nhất có thể được (thực hiện việc chia số cần kiểm tra cho số trên rãnh 2, lấy phần dư)

Xét số còn lại (số dư) :

- Nếu số còn lại là 0 thì input không là số nguyên tố (vì nó chia hết cho số trên rãnh 2)

- Nếu số còn lại khác 0 thì tăng số trên rãnh 2 thêm một đơn vị: nếu số trên rãnh 2 bằng số trên rãnh 1 (số n) thì input n là số nguyên tố vì n đã không chia hết cho bất kỳ số nào từ 2 đến n -1. Nếu số trên rãnh 2 nhỏ hơn số trên rãnh 1 thì ta lặp lại quá trình trên với số mới trên rãnh 2.



Hình 7.2 - TM với băng 3 rãnh

Hình 7.2 trên mô tả một TM với k = 3, kiểm tra số n = 47 viết trên rãnh 1 dưới dạng nhị phân, TM đang thực hiện phép chia 47 cho 5. Nó đã trừ 2 lần số 5 vào số 47, vậy ở rãnh 3 hiện đang có số 37.

Đánh dấu ký hiệu (Checking off symbols)

Kỹ thuật đánh dấu thường dùng để nhận diện các ngôn ngữ được định nghĩa bằng cách lặp lại chuỗi chẳng hạn như {ww w å*}; {wcy w, y å*, w y} hoặc {wwR w å*} hoặc các ngôn ngữ có độ dài các chuỗi con cần được so sánh, như {aibi i 1} hoặc {aibjck i = j hoặc j = k}.

Ta dùng một rãnh mở rộng trên băng để giữ ký hiệu đánh dấu . Ký hiệu xuất hiện khi ký hiệu trên rãnh ngay bên dưới nó đã hoặc đang được xét bởi TM.

Thí dụ 7.5 :Xét máy Turing M (Q, å, , , q0, B, F) nhận diện ngôn ngữ L có dạng {wcw w (a+b)+} với các thành phần như sau :

Q = {[q, d] q = q1, ..., q9 và d = a, b hoặc B} = {q1, ..., q9} {a, b, B} (thành phần thứ hai của các trạng thái dùng để lưu trữ ký hiệu nhập)

å = {[B, d] d = a, b, c} (ký hiệu nhập [B, d] được xác định bởi d)

= {[X, d] X = B hoặc ; d = a, b, c hoặc B}.

q0 = [q1, B]

B = [B, B] được định nghĩa là B, ký hiệu Blank.

F = {[q9, B]}.

Với d = a hoặc b; e = a hoặc b, ta định nghĩa hàm chuyển như sau:

1) d([q1, B], [B, d]) = ([q2, d], [Ö, d], R)

M đánh dấu ký hiệu được duyệt trên băng, lưu trữ vào bộ điều khiển và dịch chuyển sang phải.

2) d([q2, d], [B, e]) = ([q2, d], [B, e], R)

M tiếp tục dịch phải trên các ký hiệu chưa đánh dấu và tìm c.

3) d([q2, d], [B, c]) = ([q3, d], [B, c], R)

Khi tìm thấy c, M đi vào trạng thái mà thành phần đầu tiên là q3.

4) d([q3, d], [Ö, e]) = ([q3, d], [Ö, e], R)

M dịch phải qua các ký hiệu đã đánh dấu.

5) d([q3, d], [B, d]) = ([q4, B], [Ö, d], L)

M gặp ký hiệu chưa đánh dấu. Nếu ký hiệu chưa đánh dấu giống với ký hiệu đang lưu trong bộ điều khiển thì M đánh dấu rồi dịch trái. Nếu ký hiệu không giống ký hiệu lưu trong bộ điều khiển thì M không dịch chuyển nữa và không chấp nhận input. M cũng dừng nếu ở trạng thái q3 và gặp ký hiệu [B, B] trước khi gặp ký hiệu chưa đánh dấu.

6) d([q4, B], [Ö, d]) = ([q4, B], [Ö, d], L)

M dịch trái trên các ký hiệu đã đánh dấu.

7) d([q4, B], [B, c]) = ([q5, B], [B, c], L)

M gặp ký hiệu c.

8) d([q5, B], [B, d]) = ([q6, B], [B, d], L)

Nếu ký hiệu ngay bên trái c chưa được đánh dấu thì M tiến sang trái để tìm ký hiệu bên phải nhất đã được đánh dấu.

9) d([q6, B], [B, d]) = ([q6, B], [B, d], L)

M tiếp tục dịch chuyển sang trái.

10) d([q6, B], [Ö, d]) = ([q1, B], [Ö, d], R)

M gặp ký hiệu đã đánh dấu, nó dịch phải để lấy ký hiệu chưa đánh dấu bên cạnh và tiếp tục vòng lặp so sánh. Khi đó, thành phần thứ 1 lại trở thành q1.

11) d([q5, B], [Ö, d]) = ([q7, B], [Ö, d], R)

M ở trạng thái [q5, B] ngay sau khi vượt sang trái c. Nếu ký hiệu xuất hiện ngay trước c đã được đánh dấu thì tất cả các ký hiệu trước c đều đã được đánh dấu. M phải kiểm tra xem bên phải c còn có ký hiệu nào chưa được đánh dấu hay không. Nếu không còn ký hiệu nào thì M chấp nhận input.

12) d([q7, B], [B, c]) = ([q8, B], [B, c], R)

M dịch sang phải c.

13) d([q8, B], [Ö, d]) = ([q8, B], [Ö, d], R)

M tiếp tục dịch sang phải trên các ký hiệu đã được đánh dấu.

14) d([q8, B], [B, B]) = ([q9, B], [Ö, B],)

M tìm gặp Blank, nó dừng và chấp nhận chuỗi. Nếu M gặp ký hiệu chưa được đánh dấu khi thành phần thứ 1 là q8 thì nó dừng và không chấp nhận.

Dịch qua (Shifting over)

Máy Turing có thể tạo ra một không gian trống trên băng bằng cách dời các ký hiệu không trống trên băng đi sang phải hữu hạn ô. Để làm điều đó đầu đọc phải thực hiện dịch phải, lặp lại việc lưu ký hiệu đọc được vào bộ điều khiển và thay thế chúng bằng ký hiệu đọc được ở ô bên trái. Nếu có đủ ô trống, TM cũng có thể chuyển dịch một khối ký hiệu sang trái một cách tương tự.

Thí dụ 7.6 :Xây dựng TM M(Q, å, , , q0, B, F) dịch toàn bộ các ký hiệu không trống trên băng sang phải 2 ô.

Ta giả sử không có Blank giữa các ký hiệu không trống, vì vậy khi đầu đọc gặp Blank thì nó đã dịch xong các ký hiệu khác trống trên băng. Tập các trạng thái Q chứa các phần tử dạng [q, A1, A2] với q = q1 hoặc q2 và A1, A2 . Gọi X là một ký hiệu đặc biệt được chấp nhận trên băng của M, nó không được sử dụng với mục đích nào khác ngoài quá trình dịch chuyển trên băng. M bắt đầu với trạng thái [q1, B, B] và hàm chuyển thực hiện như sau:

Với Ai - {B, X}

1) d([q1, B, B], A1) = ([q1, B, A1], X, R)

M lưu ký hiệu đọc đầu tiên vào thành phần thứ 3 trong bộ điều khiển, ghi X vào ô đang đọc rồi dịch sang phải.

2) d([q1, B, A1], A2) = ([q1, A1, A2], X, R)

M chuyển ký hiệu ở thành phần thứ 3 sang thành phần thứ 2, lưu trữ ký hiệu đọc được vào thành phần thứ 3, viết X vào ô đang đọc rồi dịch sang phải.

3) d([q1, A1, A2], A3) = ([q1, A2, A3], A1, R)

Bắt đầu từ bước chuyển này, M lần lượt đọc vào một ký hiệu, ghi nó vào thành phần thứ 3, chuyển ký hiệu được ghi trước đó ở thành phần thứ 3 sang thành phần thứ 2, chép lại ký hiệu ở thành phần thứ 2 vào ô đang đọc rồi dịch sang phải.

4) d([q1, Ai - 2, Ai – 1], Ai) = ([q1, Ai - 1, Ai], Ai - 2, R)

5) d([q1, An - 1, An], B) = ([q2, An, B], An - 1, R)

Cho đến khi M gặp B, nó dốc nốt 2 ký hiệu cuối đang giữ trong bộ nhớ để bắt đầu đi vào trạng thái kết thúc.

6) d([q2, An, B], B) = ([q2, B, B], An, L)

Cuối cùng, tất cả các ký hiệu không trống trên băng đã được chuyển dịch sang phải 2 ô. Lúc đó nó sẽ được chuyển sang một trạng thái nào đó (có thể quay về trái, trở về đầu băng) để thực hiện một chức năng khác.

Chương trình con (Subroutines)

Cũng giống như một chương trình máy tính hiện đại, máy Turing có thể đóng vai trò tương tự như bất kỳ một kiểu chương trình con nào trong ngôn ngữ lập trình bao gồm thủ tục đệ qui hoặc có tham số.

Ý tưởng chung là ta viết một phần chương trình của TM như là một chương trình con. Nó sẽ được thiết kế có chứa một trạng thái khởi đầu và một trạng thái trở về, trạng thái trở về là trạng thái không có phép chuyển kế tiếp và nó sẽ đóng vai trò là trạng thái khởi đầu của một TM khác hoặc là một trạng thái nào đó trong một TM khác. Nghĩa là từ trạng thái trở về của TM này ta tiếp tục các phép chuyển của một TM khác, sự kiện này có ý nghĩa như là gọi một chương trình con khác hoặc tiếp tục thực hiện chương trình cấp trên. Lưu ý, các trạng thái của chương trình con phải phân biệt với chương trình cấp trên của nó.

Thí dụ 7.7 :Thiết kế TM thực hiện phép nhân 2 số nguyên m, n.

. Input : 0m10n

. Output : 0m   n

M bắt đầu với 0m10n trên băng và kết thúc với 0m   n trên băng được bao quanh bởi các Blank.

Ý tưởng chung là đặt thêm số 1 sau 0m10n rồi chép khối n số 0 sang phải m lần mỗi lần xoá một con 0 bên trái của 0m. Ta được kết quả cuối cùng là 10n10m   n . Bây giờ ta chỉ việc xoá 10n1 ta sẽ được kết quả 0m   n .

Phần chính của giải thuật trên là thủ tục COPY để chép n số 0 sang phải. Thủ tục này được xác định bằng các hàm chuyển sau:

	0	1	2	B
q1	(q2, 2, R)	(q4, 1, L)		
q2	(q2, 0, R)	(q2, 1, R)		(q3, 0, L)
q3	(q3, 0, L)	(q3, 1, L)	(q1, 2, R)	
q4		(q5, 1, R)	(q4, 0, L)	

Ở trạng thái q1 nhìn thấy 0, M đổi 0 thành 2 và đi vào trạng thái q2. Ở trạng thái q2, M dịch phải tới Blank viết 0 rồi dịch trái trong trạng thái q3. Khi ở trạng thái q3 mà gặp 2, M đi vào trạng thái q1 để tiếp tục lặp lại quá trình trên cho tới khi gặp 1. Trạng thái q4 được dùng để biến đổi 2 thành 0 và thủ tục dừng tại q5.

Để làm đầy đủ chương trình ta phải thêm các trạng thái để biến đổi hình thái khởi đầu q00m10n thành B0m-11q10n1. Tức là ta cần ba qui tắc:

d(q0, 0) = (q6, B, R)

d(q6, 0) = (q6, 0, R)

d(q6, 1) = (q1, 1, R)

Sau đó, ta lại thêm các phép chuyển và trạng thái cần thiết để biến đổi từ hình thái Bi0m-i1q50n10n i thành Bi+10m-i-11q10n10n   i là trạng thái bắt đầu lại việc COPY, đồng thời kiểm tra i = m hay không (khi tất cả các 0 của 0m đã bị xoá). Nếu i = m thì 10n1 bị xoá và quá trình tính toán sẽ dừng ở trạng thái q12. Các hàm chuyển bổ sung như sau :

	0	1	2	B
q5	(q7, 0, L)			
q7		(q8, 1, L)		
q8	(q9, 0, L)			(q10, B, R)
q9	(q9, 0, L)			(q0, B, R)
q10		(q11, B, R)		
q11	(q11, B, R)	(q12, B,)		

CÁC BIẾN DẠNG CỦA MÁY TURING

Sau đây, ta sẽ xét thêm một số dạng khác của máy Turing, chúng có vẻ phức tạp và tinh vi hơn, song thực tế chúng cũng đều tương đương với mô hình TM cơ bản đã định nghĩa ở trên.

Máy Turing với băng vô hạn 2 chiều

Máy Turing với băng vô hạn hai chiều cũng tương tự như mô hình gốc (TM vô hạn một chiều băng), chỉ khác là băng của nó không có cận trái như mô hình gốc, nghĩa là ta xem như TM có vô hạn Blank ở cả hai đầu băng. Vì thế hàm   được mở rộng thêm bằng cách xét thêm các trường hợp đặc biệt tại cận trái như sau :

Nếu   (q, X) = (p, Y, L) thì qX ⊢ pBY

Nếu   (q, X) = (p, B, R) thì qX ⊢ p

ĐỊNH LÝ 7.1 : Nếu L được nhận diện bởi TM với băng vô hạn hai chiều thì L cũng được nhận diện bằng TM vô hạn một chiều băng

Chứng minh

Gọi M2 là TM với băng vô hạn hai chiều M2 (Q2,   2,   2,   2, q2, B, F2) nhận diện L. Ta xây dựng M1 là TM vô hạn một chiều băng nhận diện L. Băng của M1 có 2 rãnh:

- Rãnh trên biểu diễn cho băng của M2 phía phải đầu đọc lúc khởi đầu.

- Rãnh dưới biểu diễn cho băng phía trái đầu đọc lúc khởi đầu theo thứ tự ngược lại.

...	A-5	A-4	A-3	A-2	A-1	A0	A1	A2	A3	A4	A5	...

(a) - Băng của M2

A0	A1	A2	A3	A4	A5	...
Ë	A-1	A-2	A-3	A-4	A-5	...

(b) - Băng của M1

Hình 7.3 - Băng nhập của TM M2 và M1

M1 thực hiện các phép chuyển tương tự như M2 nhưng khi M2 thực hiện các phép chuyển phía phải đầu đọc thì M1 làm việc với rãnh trên, khi M2 thực hiện các phép chuyển bên trái đầu đọc thì M1 làm việc ở rãnh dưới

Một cách hình thức M1 (Q1, 1, 1, 1, q1, B, F1), trong đó :

Q1 là tập hợp các đối tượng dạng [q, U] hoặc [q, D], trong đó q là trạng thái trong Q2, còn U, D dùng chỉ rằng M1 đang làm việc với rãnh trên (Up) hay rãnh dưới (Down). Các ký hiệu băng của M1 (các ký hiệu thuộc 1) có dạng [X, Y] trong đó X, Y thuộc 2, hơn nữa Y có thể là là ký hiệu không có trong 2 dùng để đánh dấu ô trái nhất trên băng của M1 .

1 là tập hợp các đối tượng dạng [a, B] trong đó a 2.

F1 = {[q, U], [q, D] q Î F2}.

Hàm chuyển 1 có dạng như sau:

1) 1(q1, [a, B]) = ([q, U], [X,], R) nếu 2(q2, a) = (q, X, R)

Nếu M2 chuyển sang phải trong lần chuyển đầu tiên thì M in trên rãnh dưới, ghi nhớ U vào thành phần thứ hai của trạng thái và dịch phải. Thành phần thứ nhất của trạng thái lưu trạng thái của M2. M1 in X (ký hiệu mà M2 in ra) ở rãnh trên.

1. a Î 2 U {B} :

1(q1, [a, B]) = ([q, D], [X,], R) nếu 2(q2, a) = (q, X, L)

Nếu M2 chuyển sang trái trong lần chuyển đầu tiên thì M1 in trên rãnh dưới, ghi nhớ D vào thành phần thứ hai của trạng thái và dịch phải. Thành phần thứ nhất của trạng thái lưu trạng thái của M2. M1 in X (ký hiệu mà M2 in ra) ở rãnh trên.

3) [X, Y] 1, với Y và A = L hoặc R :

1([q, U], [X, Y]) = ([p, U], [Z, Y], A) nếu 2(q, X) = (p, Z, A)

M1 ở rãnh trên thực hiện tương tự như M2.

4) 1([q, D], [X, Y]) = ([p, D], [X, Z], A) nếu 2(q, Y) = (p,Z, A)

(Trong đó nếu A = L thì A = R và nếu A = R thì A = L)

Ở rãnh dưới, M1 làm tương tự M2 nhưng dịch chuyển đầu đọc theo hướng ngược lại.

5) 1([q, U], [X,]) = 1([q, D], [X,]) = ([p, C], [Y,], R]

nếu 2(q, X) = (p, Y, A)

(Trong đó C = U nếu A = R, C = D nếu A = L)

M1 làm tương tự M2 ở ô khởi đầu, công việc tiếp theo của M1 thực hiện ở rãnh trên hay dưới phụ thuộc vào hướng chuyển đầu đọc của M2.

Máy Turing với nhiều băng vô hạn hai chiều

Xét máy Turing có một bộ điều khiển có k đầu đọc và k băng vô hạn hai chiều. Mỗi phép chuyển của máy Turing, phụ thuộc vào trạng thái của bộ điều khiển và ký tự đọc được tại mỗi đầu đọc, nó có thể thực hiện các bước sau :

1) Chuyển trạng thái.

2) In ký hiệu mới tại mỗi đầu đọc để thay thế ký hiệu vừa đọc.

3) Đầu đọc có thể giữ nguyên vị trí hoặc dịch trái hoặc dịch phải 1 ô một cách độc lập nhau.

Khởi đầu input xuất hiện trên băng thứ nhất, các băng khác chỉ toàn Blank.

Một máy Turing như vậy gọi là máy Turing với nhiều băng vô hạn hai chiều.

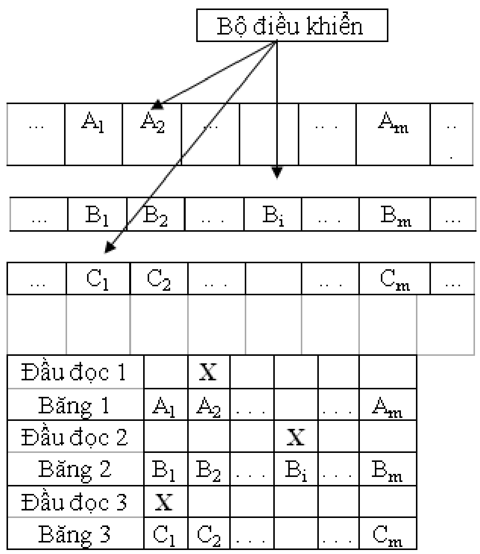
ĐỊNH LÝ 7.2 : Nếu L được nhận dạng bởi máy Turing nhiều băng vô hạn hai chiều thì nó cũng được nhận dạng bởi máy Turing một băng vô hạn hai chiều.

Chứng minh

Giả sử L được nhận diện bởi máy Turing k băng vô hạn hai chiều M1, ta xây dựng máy Turing M2 một băng với 2k rãnh, 2 rãnh sẽ mô phỏng một băng của M1 bằng cách: một rãnh giữ ký hiệu trên băng của M1 một rãnh kia giữ ký hiệu đánh dấu vị trí đầu đọc.

Mỗi phép chuyển của M1 được mô phỏng bằng M2 như sau:

M2 xuất phát tại vị trí trái nhất chứa ký hiệu đánh dấu đầu đọc, M2 quét sang phải, khi qua mỗi ô có đánh dấu vị trí đầu đọc nó ghi nhớ ký hiệu tại vị trí này và đếm số vị trí đầu đọc đã gặp. Khi M2 đi sang phải và đã đếm đủ k đầu đọc thì nó đã có đủ thông tin để xác định phép chuyển tương tự như M1, M2 lại quét từ phải sang trái, khi đi ngang qua mỗi ô có đánh dấu đầu đọc nó in ký hiệu thay thế ký hiệu tại đầu đọc (như M1) chuyển vị trí đánh dấu đầu đọc (như M1 chuyển đầu đọc của nó). Cuối cùng M2 đổi trạng thái trong bộ điều khiển của nó thành trạng thái mà M1 chuyển tới.



Hình 7.4 - Máy Turing 1 băng mô phỏng máy Turing 3 băng

Thí dụ 7..8 :Ngôn ngữ {ww w   (0+1)*} có thể được chấp nhận bởi một máy Turing có 2 băng bằng cách như sau: Đầu tiên, nó chép lại chuỗi nhập ở băng thứ nhất lên băng thứ hai. Sau đó, trên băng thứ nhất đầu đọc chuyển dần từ cận trái sang cận phải của chuỗi, trong khi trên băng thứ hai đầu đọc lại chuyển ngược lại từ cận phải sang cận trái của chuỗi đó. Chuỗi được chấp nhận nếu suốt quá trình đó, các ký hiệu đọc được trên 2 băng luôn luôn đồng nhất.

Như ta đã biết, để đoán nhận ngôn ngữ này bằng TM một băng thì đầu đọc phải dịch chuyển tới lui rất nhiều lần để so sánh hai nửa của chuỗi nhập từ cả hai đầu băng. Như vậy, số bước dịch chuyển của nó xấp xỉ bằng bình phương độ dài chuỗi nhập, trong khi TM với 2 băng nhập chỉ cần thực hiện số bước chuyển tỷ lệ với độ dài của chuỗi nhập.

Máy Turing không đơn định

Máy Turing không đơn định có mô hình tương tự như mô hình gốc nhưng điểm khác biệt ở chỗ là trong mỗi lần chuyển, máy Turing có thể lựa chọn một trong một số hữu hạn các trạng thái kế tiếp, lựa chọn hướng chuyển đầu đọc, và lựa chọn ký hiệu in ra trên băng để thay thế ký hiệu vừa đọc được. Máy Turing trong mô hình gốc còn gọi là máy Turing đơn định.

ĐỊNH LÝ 7.3 : Nếu L được chấp nhận bởi máy Turing không đơn định M1 thì L cũng được chấp nhận bởi một máy Turing đơn định M2 nào đó.

Chứng minh

Với một trạng thái và một ký hiệu băng bất kỳ của M1, có một số hữu hạn các phép chuyển đến trạng thái kế tiếp, ta có thể đánh số các trạng thái này là 1, 2, ... Gọi r là số lớn nhất của số các cách lựa chọn với một cặp trạng thái và ký kiệu bất kỳ. Ta có, mọi dãy các phép chuyển trạng thái đều được chỉ ra bằng một dãy chứa các số từ 1 đến r. Ngược lại một dãy hữu hạn bất kỳ gồm các số từ 1 đến r có thể biểu diễn cho một dãy các phép chuyển nào đó cũng có thể không. M2 được thiết kế có ba băng:

Băng 1 chứa input

Băng 2 sinh ra dãy chứa các số từ 1 đến r một cách tự động theo tính chất dãy ngắn sinh ra trước, nếu các dãy cùng độ dài thì nó sinh ra theo thứ tự liệt kê số (numerical order).

Băng 3 dùng chép input trên băng 1 vào để xử lý: với mỗi số sinh ra trên băng 2, M2 chép input trên băng 1 vào băng 3 và thực hiện các phép chuyển theo dãy số trên băng 2.

Nếu có một chuỗi nào đó trên băng 2 làm cho M2 đi vào trạng thái kết thúc thì M2 dừng và chấp nhận input. Nếu không có chuỗi nào như vậy thì M2 không chấp nhận input. Tất nhiên M2 chấp nhận input khi và chỉ khi M1 chấp nhận input.

Máy Turing nhiều chiều

Máy Turing nhiều chiều gồm một bộ điều khiển hữu hạn, nhưng băng của nó là một mảng k chiều vô hạn về cả 2k phía. Với một số k nào đó, phụ thuộc vào trạng thái và một ký hiệu được đọc, máy thay đổi trạng thái, in một ký hiệu mới tại ô đang đọc và dịch chuyển đầu đọc theo một trong 2k phía.

ĐỊNH LÝ 7.4: Nếu L được chấp nhận bởi máy Turing k chiều M1 thì L cũng được chấp nhận bởi một máy Turing một chiều M2 nào đó.

(Phần chứng minh, xem như bài tập)

Máy Turing nhiều đầu đọc

Máy Turing nhiều đầu đọc có k đầu đọc được đánh số từ 1 đến k với k là một số hữu hạn nào đó, nhưng chỉ có một băng input. Một phép chuyển của máy Turing phụ thuộc vào trạng thái và các ký tự được đọc bởi mỗi đầu băng. Mỗi đầu dịch chuyển một cách độc lập sang trái, sang phải hoặc đứng yên.

ĐỊNH LÝ 7.5 : Nếu L được chấp nhận bởi máy Turing k đầu đọc M1 thì L cũng được chấp nhận bởi một máy Turing một đầu đọc M2 nào đó.

(Phần chứng minh, xem như bài tập)

GIẢ THUYẾT CHURCH

Giả thuyết rằng khái niệm trực giác “Hàm tính được” (computable function) có thể được định nghĩa bằng lớp các hàm đệ quy bộ phận là giả thuyết Church hay còn được gọi là luận đề Church - Turing.

Trong khi chúng ta không thể hy vọng để chứng minh giả thuyết Church cũng như những định nghĩa không hình thức về "sự tính được", chúng ta có thể cho những dẫn chứng về những khả triển của chúng. Trong một thời gian dài, khái niệm trực giác về "sự tính được" đặt không giới hạn trên số bước hoặc tổng số các lưu trữ, có vẻ như các hàm đệ quy bộ phận thì có thể tính được một cách trực giác mặc dù cũng có một số hàm không thể tính được trừ khi ta đặt giới hạn cho việc tính toán sau đó hoặc ít nhất thiết lập được liệu có hay không có phép tính cuối cùng.

Điều còn không rõ là liệu lớp các hàm đệ quy bộ phận có thể bao hàm tất cả mọi "hàm tính được". Những nhà logic học đã đưa ra nhiều công thức khác, chẳng hạn như phép tính- , hệ thống Post và các hàm đệ quy tổng quát. Tất cả chúng được định nghĩa cùng một lớp hàm, cụ thể là hàm đệ quy bộ phận. Hơn nữa, các mô hình máy tính trừu tượng, chẳng hạn như mô hình RAM (Random Access Machine) cũng được xem xét như một hàm đệ quy bộ phận.

Mô hình RAM bao gồm một số vô hạn các từ nhớ, đánh số 0, 1, ..., mỗi một từ nhớ có thể lưu giữ một số nguyên bất kỳ và một số hữu hạn các thanh ghi số học cũng có khả năng giữ các số nguyên bất kỳ. Các số nguyên có thể được giải mã thành các dạng thông thường của các chỉ thị máy tính. Chúng ta sẽ không định nghĩa mô hình RAM một cách hình thức hơn, nhưng sẽ rõ ràng hơn nếu chúng ta chọn một tập các chỉ thị phù hợp, RAM sẽ mô phỏng mọi máy tính hiện có. Chứng minh rằng mô hình máy Turing cũng có khả năng tương đương như mô hình RAM được chỉ ra dưới đây hay có thể nói một máy Turing cũng có tác dụng như một kiểu RAM.

Mô phỏng mô hình RAM bởi máy Turing

ĐỊNH LÝ 7.6: Một máy Turing có thể mô phỏng một RAM, với điều kiện là mỗi chỉ thị RAM cũng có thể được mô phỏng bởi một TM.

Chứng minh

Ta sử dụng một TM M nhiều băng để thực hiện quá trình mô phỏng. Một băng của M giữ các từ của RAM được cho bởi các giá trị như dưới đây. Băng có dạng sau :

0*v0 # 1*v1 # 10*v2 # …# i*vi # …

trong đó vi là nội dung băng viết dưới dạng nhị phân của từ thứ i. Tại mỗi thời điểm, sẽ có một số hữu hạn các từ của RAM có thể được dùng và M chỉ cần lưu giữ lại các giá trị cho đến khi có một số lượng từ lớn nhất được sử dụng sau đó.

RAM có một số hữu hạn các thanh ghi số học. M dùng một băng để giữ nội dung của mỗi thanh ghi này, một băng để giữ số đếm vị trí (location counter), nơi chứa số thứ tự các từ mà chỉ thị kế tiếp sẽ gọi đến. Và một băng nữa dùng như là thanh ghi địa chỉ bộ nhớ (memory address register) trong đó lưu giữ số của từ nhớ.

Giả sử rằng 10 bit đầu tiên của một chỉ thị biểu thị một toán tử chuẩn của máy tính, chẳng hạn như LOAD, STORE, ADD, … và những bit sau đó xác định địa chỉ của một toán hạng. Trong khi chúng ta sẽ xem xét một cách chi tiết việc cài đặt tất cả các chỉ thị máy tính chuẩn, một ví dụ minh họa sẽ cho thấy điều này rõ ràng hơn. Giả sử băng số đếm vị trí của M giữ số i trong hệ nhị phân. M duyệt băng này đầu tiên từ bên trái và tìm thấy # i*. Nếu một khoảng trống được đếm trước khi tìm thấy # i*, có nghĩa là không có chỉ thị nào trong từ i, vì thế RAM và M ngừng lại. Nếu # i* được tìm thấy, chuỗi bit theo sau ký hiệu * cho đến ký hiệu # sau đó sẽ được xem xét. Giả sử 10 bit đầu tiên là mã lệnh của "ADD to register 2" và phần chuỗi bit còn lại là một số j trong hệ nhị phân. M thêm 1 vào i trên băng số đếm vị trí và sao chép j vào băng địa chỉ bộ nhớ. Sau đó, M lại tìm kiếm # j* trên băng đầu tiên,

một lần nữa lại bắt đầu từ bên trái (chú ý rằng # 0* đánh dấu vị trí cận trái). Nếu # j* không tìm thấy, ta giả sử từ j giữ 0 và đi tiếp đến chỉ thị kế tiếp của RAM. Nếu # j* vj# được tìm thấy, vj sẽ đựoc thêm vào nội dung của thanh ghi 2, nơi chứa chính nó trên băng. Tiếp tục lặp lại vòng lặp này với chỉ thị kế tiếp.

Lưu ý rằng trong giải thuật này, mặc dù mô phỏng RAM dùng một máy Turing nhiều băng, nhưng theo Định lý 7.2, nếu ta dùng TM với một băng vô hạn hai chiều cũng sẽ thành công song sẽ phức tạp hơn.

MÁY TURING NHƯ LÀ MỘT BỘ LIỆT KÊ

Ta đã xét máy Turing như là một máy dùng nhận dạng ngôn ngữ và tính toán các hàm. Một quan điểm rất có ích nữa là xem máy Turing như là bộ liệt kê, tức là nó có khả năng sinh ra ngôn ngữ.

Xét máy Turing có nhiều băng, một trong các băng đó được xem là băng output, trên băng này một ký hiệu được viết lên sẽ không bị thay đổi và đầu đọc của băng này không bao giờ dịch trái.

Giả sử trên băng output, M sẽ viết chuỗi các ký tự thuộc bộ chữ cái , các chuỗi được viết ngăn cách nhau bởi dấu #. Ta định nghĩa G(M) là ngôn ngữ sinh bởi máy Turing M, là tập hợp tất cả các từ w * được viết giữa hai dấu # trên băng output. Chú ý rằng trừ khi M không dừng, G(M) luôn luôn hữu hạn. Ta cũng không yêu cầu các từ được sinh ra theo một thứ tự nào đó, và cũng không yêu cầu mỗi từ chỉ sinh ra đúng một lần.

Tính chất đệ qui liệt kê của tập sinh

BỔ ĐỀ 7.1: Nếu L là G(M1) với TM M1 nào đó thì L là tập đệ qui liệt kê

Chứng minh

Ta xây dựng M2 có nhiều hơn M1 một băng, M2 sẽ thực hiện tương tự M1, M2 dùng tất cả các thành phần của M1 chỉ trừ băng input của M1, nhưng khi M1 in # trên băng output của M1 thì M2 lấy input của M2 so sánh với từ vừa được sinh trên băng output của M1. Nếu giống thì M2 chấp nhận, ngược lại M2 tiếp tục làm tương tự M1. Rõ ràng M2 chấp nhận input w nếu và chỉ nếu M1 sinh ra w, vậy G(M1) là tập đệ qui liệt kê.

Chứng minh điều ngược lại của bổ đề trên là khó khăn hơn. Giả sử M1 là bộ nhận dạng của một tập hợp đệ qui liệt kê L * nào đó. Nếu ta cố gắng thiết kế một bộ sinh ra L có thể sinh mọi từ thuộc * theo thứ tự nào đó là w1, w2, ... , ta cho chạy M1 trên w1, nếu M1 chấp nhận thì sinh ra w1. Rồi chạy M1 với w2, ..., cứ lần lượt như thế với mọi từ. Phương pháp này chỉ đúng nếu M1 dừng trên mọi input. Tuy nhiên, do có các tập đệ qui liệt kê nhưng không đệ qui vì vậy M1 có thể không dừng với một input wi nào đó và tất nhiên ta sẽ không bao giờ xét được các từ sau đó wi+1, wi+2, ... ngay cả khi M1 chấp nhận chúng.

Từ phân tích trên, ta thấy vấn đề đặt ra là phải thiết kế bộ sinh sao cho nó có thể tránh được trường hợp trên. Để làm như vậy trước hết ta đánh số thứ tự các từ thuộc * rồi ta sinh ra từng cặp số nguyên dương (i, j). Việc sinh ra cặp (i, j) có ý nghĩa như là M1 sinh ra từ thứ i bằng j bước. Cụ thể, ta đánh số các từ trong * theo "thứ tự chuẩn" (canonical order) như sau: liệt kê các từ theo độ dài, với các từ có cùng độ dài được liệt kê theo thứ tự số, tức là nếu = {a0, a1, ..., ak-1} thì mỗi từ w * coi

như là một số trong hệ k-phân. Ta có thể thiết kế TM sinh ra các từ theo thứ tự chuẩn là không khó khăn gì.

Thí dụ 7.9 :Nếu   = {0,1} thì các từ liệt kê theo thứ tự chuẩn là: , 0, 1, 00, 01, 10, 11, 000, 001, ... Xét cách sinh ra cặp sinh (i, j) sau một lượng thời gian hữu hạn. Nếu sinh theo thứ tự (1, 1), (1, 2), ... thì sẽ không bao giờ sinh được cặp (i, j) với i > 1. Thay vào đó ta cho sinh ra cặp (i, j) theo thứ tự i + j, vì số lượng cặp (i, j) thỏa i + j bằng hằng số là hữu hạn nên cặp (i, j) bất kỳ sẽ được sinh ra sau một lượng thời gian hữu hạn, cụ thể ta có thể chứng minh: cặp (i, j) sẽ được sinh ở lần sinh thứ :

(i + j - 1)(i + j - 2) / 2+ i.

Máy Turing sinh ra các cặp sinh (i, j) viết trong hệ nhị phân là dễ dàng được thiết kế và ta gọi máy Turing này là bộ sinh cặp.

ĐỊNH LÝ 7.7 : Một ngôn ngữ là tập đệ qui liệt kê nếu và chỉ nếu nó là G(M2) với TM M2 nào đó.

Chứng minh

Với bổ đề 1 ta chỉ cần chỉ ra rằng nếu L = L(M1) thì L được sinh ra bởi TM M2. M2 tương tự như bộ sinh cặp. Khi (i, j) được sinh ra, M2 sản xuất từ thứ i theo thứ tự chuẩn và làm tương tự M1 trên từ wi với j bước. Nếu M1 chấp nhận từ thứ i với j bước thì M2 sinh ra wi. Chắc chắn rằng M2 không sinh ra từ không thuộc L, đặt w là từ thứ i trong thứ tự chuẩn trên bộ chữ cái L và M1 chấp nhận w bằng j bước. Vì chỉ cần một lượng thời gian hữu hạn để M2 sinh ra bất kỳ từ nào và làm tương tự như M1 nên M2 chắc chắn sinh ra (i, j). Lúc này w sẽ được sinh ra bởi M2. Vậy L = G(M2)

HỆ QUẢ : Nếu L là tập đệ qui liệt kê thì có một bộ sinh sinh ra mỗi từ trong L đúng một lần.

Tính chất đệ qui của tập sinh

BỔ ĐỀ 7.2: Nếu L là tập đệ qui thì có một bộ sinh in ra các từ của L theo thứ tự chuẩn và không in ra các từ khác.

Chứng minh

Đặt L = L(M1)   * trong đó M1 dừng với mọi input. Ta xây dựng M2 sinh ra L như sau M2 sinh ra các từ thuộc   * mỗi lần một từ theo thứ tự chuẩn. Sau khi sinh ra một từ w, M2 làm tương tự M1 trên w. Nếu M1 chấp nhận w thì M2 sinh ra w. Vì M1 chắc chắn dừng nên M2 cũng sẽ dừng sau hữu hạn bước và chắc chắn sẽ xét mỗi từ thuộc   *. Vậy M2 sinh ra L theo thứ tự chuẩn.

Điều ngược lại của bổ đề trên cũng đúng, tức là, nếu L được sinh ra theo thứ tự chuẩn thì L là tập đệ qui. Nghĩa là có TM nhận diện M tồn tại, tuy nhiên không có một giải thuật cụ thể cho TM này.

Giả sử M1 sinh ra L theo thứ tự chuẩn. Một ý nghĩ tự nhiên là ta xây dựng M2 làm tương tự M1 trên input w cho tới khi M1 sinh ra w hoặc sinh ra từ sau w (theo thứ tự chuẩn). Trong trường hợp đầu, M2 chấp nhận w, trong trường hợp sau M2 dừng nhưng không chấp nhận w. Rõ ràng nếu L hữu hạn thì M1 có thể không dừng sau khi sinh ra từ cuối cùng trong L (vì theo định lý trên M1 không sinh từ nào không thuộc L). Trong trường hợp này M2 cũng không dừng. Điều này chỉ gặp khi L hữu hạn, nhưng do không có cách xác định TM có sinh ra tập hữu hạn hay không hoặc nếu biết TM sinh ra tập hữu hạn thì cũng không thể biết tập hợp đó là gì. Vậy ta biết là có TM chấp nhận L, nhưng không thể đưa ra một giải thuật cụ thể cho TM này.

ĐỊNH LÝ 7.8 : L là tập đệ qui nếu và chỉ nếu L được sinh ra theo thứ tự chuẩn.

Chứng minh

Ta chỉ cần chứng minh phần nếu.

Khi L hữu hạn thì sẽ có một ôtômát chấp nhận L và vì vậy L được chấp nhận bởi TM luôn luôn dừng trên tất cả các input.

SỰ TƯƠNG ĐƯƠNG GIỮA VĂN PHẠM kIỂU 0 VÀ MÁY TURING

Họ văn phạm rộng lớn nhất theo sự phân cấp của Noam Chomsky đòi hỏi các luật sinh có dạng , với , là các chuỗi tùy ý chứa ký hiệu văn phạm sao cho . Lớp văn phạm này được biết như là văn phạm kiểu 0, văn phạm ngữ cấu hay văn phạm không hạn chế.

Thí dụ 7.10 :Cho một văn phạm không hạn chế sinh ra ngôn ngữ

L = { ai i là lũy thừa dương của 2 } với tập luật sinh như sau :

G ({S, A, B, C, D, E}, {a, }, P, S)

Với P = {1. S ACaB

2. Ca aaC

3. CB DB

4. CB E

5. aD Da

6. AD AC

7. aE Ea

8. AE }

Trong văn phạm trên, biến A và B giữ vai trò là ký hiệu đánh dấu cận trái và cận phải của một chuỗi thuộc ngôn ngữ. C di chuyển từ trái sang phải qua chuỗi các ký hiệu a nằm giữa hai biến A và B, và gấp đôi số ký hiệu a đó lên theo luật sinh (2). Khi C chạm đến cận phải B, nó sẽ thay thế thành D hay E theo luật sinh (3) hoặc (4). Nếu D được chọn thay thế thì D lại quay về trái theo luật sinh (5), cho đến khi gặp cận trái A thì thay thế lại thành C theo luật sinh (6) và cho phép lặp lại chu trình. Còn nếu E được chọn để thay thế, thì theo luật sinh (4), B sẽ biến mất, sau đó E quay về trái theo luật sinh (7) cho đến khi gặp cận trái A thì xóa A và mất đi theo luật sinh (8), cho ra chuỗi có dạng 2i ký hiệu a, với i > 0.

Có thể chứng minh bằng quy nạp theo số bước dẫn xuất rằng nếu luật sinh (4) chưa được dùng đến thì chuỗi trong dẫn xuất có một trong ba dạng như sau :

1. S
2. AaiCajB, với i + 2j là một lũy thừa dương của 2.
3. AaiDajB, với i + j là một lũy thừa dương của 2.

Khi luật sinh (4) được áp dụng thì ta sẽ có chuỗi dạng AaiE, trong đó i là một lũy thừa dương của 2. Sau đó, ta chỉ có thể áp dụng i lần luật sinh (7) để đi tới dạng câu AEai. Cuối cùng, với luật sinh (8), ta thu được chuỗi dạng ai với i là lũy thừa dương của 2.

Phần tiếp theo dưới đây, chúng ta sẽ xét mối tương quan giữa văn phạm không hạn chế này và mô hình máy Turing. Chúng ta chứng minh hai Định lý dưới đây thể hiện mối tương quan giữa lớp văn phạm không hạn chế và lớp ngôn ngữ đệ quy liệt kê r.e – lớp ngôn ngữ được chấp nhận bởi một máy Turing. Định lý đầu tiên sẽ chứng tỏ rằng mọi ngôn ngữ kiểu 0 phát sinh một tập r.e. Và sau đó ta sẽ xây dựng một giải thuật để liệt kê tất cả các chuỗi thuộc văn phạm kiểu 0.

ĐỊNH LÝ 7.9 : Nếu L là L(G) với một văn phạm không hạn chế G(V, T, P, S) thì L là ngôn ngữ đệ quy liệt kê.

Chứng minh

Thiết lập một máy Turing M không đơn định, hai băng chấp nhận ngôn ngữ L. Băng thứ nhất (băng 1) của TM chứa chuỗi nhập w, còn băng thứ hai (băng 2), máy phát sinh các dạng chuỗi của G. Đầu tiên, chuỗi được phát sinh trên băng 2 là ký hiệu bắt đầu S. Sau đó, TM lặp lại quá trình sau :

(i) Chọn một cách ngẫu nhiên một vị trí i trên với 1 i , nghĩa là TM xuất phát từ bên trái chuỗi rồi tùy chọn giữa hai khả năng : hoặc chọn i là vị trí hiện tại, hoặc dịch chuyển sang phải và lặp lại quá trình.

(ii) Chọn một luật sinh trong số các luật sinh thuộc tập luật sinh của G.

(iii) Nếu chuỗi con xuất hiện trong kể từ vị trí thứ i, TM thay thế chuỗi bởi (dĩ nhiên nếu thì phải dịch chuyển phần cuối của để đủ chỗ trống cần cho phép thay thế)

(iv) So sánh chuỗi phát sinh được với chuỗi nhập w trên băng 1. Nếu giống nhau thì chuỗi mới phát sinh sẽ được chấp nhận. Nếu khác nhau thì TM trở về bước (i). Ta có thể chứng minh được rằng tất cả và chỉ có những chuỗi thuộc G mới xuất hiện trên băng 2 ở bước (iv).

Vậy L(M) = L(G) = L.

ĐỊNH LÝ 7.10 : Nếu L là ngôn ngữ đệ quy liệt kê thì L = L(G) với một văn phạm không hạn chế G nào đó.

Chứng minh

Giả sử ngôn ngữ L được chấp nhận bởi máy Turing M (Q, å, , , q0, B, F). Ta sẽ xây dựng một văn phạm không hạn chế G mà mỗi chuỗi dẫn xuất của nó phát sinh theo ba bước như sau :

(i) G phát sinh một cách ngẫu nhiên một chuỗi w thuộc . Chuỗi này được viết thành hai bản : một sẽ lưu giữ cho đến khi kết thúc, một sẽ thay đổi trong quá trình làm việc của TM.

(ii) G mô phỏng lại quá trình làm việc của của TM trên chuỗi w, bằng cách lặp lại đúng quá trình làm việc của TM.

(iii) Khi bước (ii) kết thúc, với sự xuất hiện của một trạng thái kết thúc q F của TM (nghĩa là chuỗi w đã được TM chấp nhận). Lúc đó G tiếp tục thu giảm để chuyển dạng câu đã có về như chuỗi w. Và như vậy, có nghĩa là chuỗi w đã được G sinh ra.

Một cách hình thức, ta thiết lập văn phạm G (V,   , P, S1)

Với V = (({   })   )   { S1, S2, # })

Và tập luật sinh P được xây dựng như sau :

1. a) S1   #q0 S2#

b) S2   [a, a] S2#,   a

c) S2

- Nếu   (q, X) = (p, Y, R) với p, q   ; X, Y   thì thêm các luật sinh dạng (2a) và (2b) sau đây vào tập luật sinh P :

2. a) q[a, X][b, Z]   [a, Y]p[b, Z],   a, b   { } và   Z

b) q[a, X]#   [a, Y]p[, B],   a   { }

- Nếu   (q, X) = (p, Y, L) với p, q   ; X, Y   thì thêm các luật sinh dạng (2c) sau đây vào tập luật sinh P :

c) [b, Z]q[a, X]   q[b, Z]p[a, Y],   a, b   { } và   Z

- Nếu q   F thì thêm các luật sinh (3a-e) sau đây vào tập luật sinh P:

3. a) [a, X]q   qap,   a   { } và   X

b) q[a, X]   qap,   a   { } và   X

c) q#

d) #q

e) q

Dùng các luật sinh (1a-c), ta có chuỗi dẫn xuất :

S1   G* #q0 [a1, a1][a2, a2] … [an, an]#

Chuỗi dẫn xuất này thể hiện hình thái bắt đầu của TM là : #q0a1a2 … an#. Bắt đầu từ bước này các quy tắc (2a-c) được áp dụng. Lưu ý rằng các luật sinh này trong G phản ánh các quy tắc chuyển trạng thái đã được thiết kế cho TM. Cho nên quá trình dẫn xuất lại trong G sẽ mô phỏng lại các bước chuyển hình thái trong quá trình làm việc của TM. Nếu quá trình đó chuyển đến một trong những trạng thái kết thúc q   F, tương ứng với trường hợp TM chấp nhận chuỗi a1a2 … an, thì trong văn phạm G các quy tắc (3a-e) sẽ được áp dụng tiếp theo và cho phép G dẫn xuất ra chính chuỗi nhập a1a2 … an. Hay ta có : S   G* a1a2 … an

Phần chứng minh L(M)   L(G) và L(G)   L(M) xem như bài tập.

Tổng kết chương VII: Với sự giới thiệu mô hình máy Turing như là một mô hình của sự tính toán, người ta còn đi tới khái niệm về độ phức tạp của tính toán hay “độ khó” của các bài toán. Nghiên cứu về độ phức tạp của tính toán là một hướng nghiên cứu hiện đại trong Tin học, nó có ý nghĩa lớn lao

về lý thuyết cũng như thực hành. Kết thúc chương này, sự phân lớp ngôn ngữ theo nguyên tắc của Noam Chomsky đã được thể hiện tương đối rõ ràng.

BÀI TẬP CHƯƠNG VII

7.1. Thiết kế máy Turing nhận diện ngôn ngữ:

a) { 0n 1n 0n | n ³ 1}

b) {ww R | w Î (0+1)*}

c) Tập hợp các chuỗi chứa 0 và 1, có số số 0 và số số 1 bằng nhau.

7.2. Thiết kế máy Turing tính các hàm số nguyên:

a) f(n) = n2

b) f(n) = 2n

c) f(n) = n !

7.3. Xây dựng văn phạm không hạn chế (loại 0) sinh ra các ngôn ngữ sau:

a) { ww | w Î (0+1)*}

b) { 0k | k = i2 và i ³ 1}

c) { 0i | i không là số nguyên tố}

BÀI TẬP LẬP TRÌNH

7.4. Viết chương trình máy tính mô phỏng hoạt động của các TM thiết kế trong bài tập 7.1 và 7.2.